Title: Algorytmy metaheurystyczne w kryptoanalizie szyfrów strumieniowych

Author: Iwona Polak

Uniwersytet Śląski
Wydział Informatyki i Nauki o Materiałach
Instytut Informatyki

Rozprawa doktorska

Iwona Polak

Algorytmy metaheurystyczne w kryptoanalizie szyfrów strumieniowych

Promotor: dr hab. inż. Mariusz Boryczka

Sosnowiec 2018

Spis treści

Spis symboli i skrótów

$\oplus$	XOR, alternatywa wykluczająca
$()$	permutacja identycznościowa
(b, c)	krawędź pomiędzy wierzchołkami b i c
$+$	dodawanie $\mathrm{mod}\ 256$
$śr.$	średnia arytmetyczna
α	ważność śladu feromonowego
β	ważność informacji heurystycznej
η	informacja heurystyczna pochodząca z analizowanego problemu
κ	strumień szyfrujący (*keystream*)
κ'	strumień szyfrujący z aktualnego rozwiązania
$\mathcal{MM}$AS	*MAX-MIN Ant System* – system mrówkowy MAX-MIN
ρ	współczynnik wyparowywania feromonu
τ	ślad feromonowy
τ_0	ślad feromonowy domyślny (początkowy, inicjalny)
τ_{max}	ślad feromonowy maksymalny
τ_{min}	ślad feromonowy minimalny
TABU	lista tabu
A	zbiór mrówek
a	wierzchołek

b	wierzchołek
C	szyfrogram (*ciphertext*)
c	wierzchołek
D	proces deszyfrowania (*decryption*)
E	proces szyfrowania (*encryption*)
f_{fit}	funkcja dopasowania (oceny, przystosowania)
$i \ll x$ ($i \gg x$)	przesunięcie bitowe o x w lewo (prawo)
I	liczba iteracji
i	indeks 8-bitowy całkowitoliczbowy
$iBest$	najlepsza permutacja w iteracji
j	indeks 8-bitowy całkowitoliczbowy
K	klucz (*key*)
k	mrówka
K_{PR}	klucz prywatny
K_{PU}	klucz publiczny
L	długość wiadomości w bajtach
l	numer iteracji
$M(\cdot)$	złożoność pamięciowa
max	wartość maksymalna
min	wartość minimalna
N	liczba miast
P	tekst jawny (*plaintext*)
p	prawdopodobieństwo
q_0	współczynnik względnej zależności między eksploatacją a eksploracją
r	bieżące rozwiązanie

$R(r)$	otoczenie (sąsiedztwo) rozwiązania
R^2	współczynnik determinacji
$rBest$	najlepsze znalezione rozwiązanie w całym wykonaniu algorytmu
S	tablica bajtowa zawierająca permutację liczb całkowitych $\{0, 1, ..., S-1\}$
s	odchylenie standardowe
t	czas
$T(\cdot)$	złożoność obliczeniowa (czasowa)
$tabu$	lista odwiedzonych wierzchołków
$\lvert \cdot \rvert$	moc zbioru (liczba elementów w tym zbiorze)
ACO	*Ant Colony Optimization* – algorytm optymalizacji mrowiskowej
ACS	*Ant Colony System* – system mrowiskowy
AES	*Advanced Encryption Standard* – symetryczny szyfr blokowy
AGA	*Adaptive GA* – adaptacyjny algorytm genetyczny
ANN	*Artificial Neural Network* – sztuczna sieć neuronowa
AS	*Ant System* – system mrówkowy
AS_{rank}	*rank-based Ant System* – system mrówkowy z rankingiem
BA	*Bees Algorithm* – algorytm pszczeli
BFO	*Bacterial Foraging Optimization* – optymalizacja żerowania bakterii
BWAS	*Best-Worst Ant System* – system mrówkowy „najlepszy-najgorszy”
CA	*Cellular Automaton* – automat komórkowy
CCA	*chosen ciphertext attack* – atak z wybranym szyfrogramem
COA	*ciphertext-only attack* – atak z szyfrogramem
CPA	*chosen plaintext attack* – atak z wybranym tekstem jawnym
CS	*Cuckoo Search* – system kukułczy
DC	*differential cryptanalysis* – kryptoanaliza różnicowa

DE	*Differential Evolution* – ewolucja różnicowa
DES	*Data Encryption Standard* – symetryczny szyfr blokowy
DLP	*Discrete Logarithm Problem* – problem logarytmu dyskretnego
DSA	*Digital Signature Algorithm* – asymetryczny algorytm stworzony przez NIST
DSS	*Digital Signature Standard* – standard NIST dla podpisów cyfrowych
ECC	*Elliptic Curve Cryptography* – kryptografia krzywych eliptycznych
FA	*Firefly Algorithm* – algorytm świetlika
FEAL	*Fast Data Encipherment Algorithm* – szyfr blokowy
GA	*Genetic Algorithm* – algorytm genetyczny
HC	*hill climbing* – metoda wspinaczkowa
IP	*initial permutation* – permutacja początkowa
IP^{-1}	*final permutation* – permutacja końcowa
IV	*initialization vector* – wektor inicjacyjny, wektor początkowy
KPA	*known plaintext attack* – atak ze znanym tekstem jawnym
KSA	*Key Scheduling Algorithm* – algorytm dystrybucji klucza
MA	*Memetic Algorithm* – algorytm memetyczny
ML	*Machine Learning* – uczenie maszynowe
NIST	*National Institute of Standards and Technology* – Narodowy Instytut Standaryzacji i Technologii
OTP	*one-time pad* – szyfr z kluczem jednorazowym
PRGA	*Pseudo-Random Generation Algorithm* – algorytm generowania pseudolosowego strumienia
PSO	*Particle Swarm Optimization* – optymalizacja stadna cząsteczek
RC4	*Rivest-Cipher 4* – szyfr strumieniowy
RC4+_10	szyfr RC4+ o rozmiarze 10 elementów

RC4+_16	szyfr RC4+ o rozmiarze 16 elementów
RC4_10	szyfr RC4 o rozmiarze 10 elementów
RC4_16	szyfr RC4 o rozmiarze 16 elementów
RSA	Rivest, Shamir, Adelman – szyfr asymetryczny
SA	*Simulated Annealing* – symulowane wyżarzanie
SCA	*side-channel attack* – atak kanałem bocznym
SDES	*Simplified DES* – uproszczona wersja algorytmu DES
SPN	*substitution-permutation network* – sieć podstawieniowo-permutacyjna
TEA	*Tiny Encryption Algorithm* – szyfr blokowy
TS	*Tabu Search* – przeszukiwanie z tabu
VMPC	*Variably Modified Permutation Composition* – szyfr strumieniowy
VMPC_10	szyfr VMPC o rozmiarze 10 elementów
VMPC_16	szyfr VMPC o rozmiarze 16 elementów
XOR	*eXclusive OR* – alternatywa wykluczająca
XTEA	*eXtended TEA* – rozszerzona wersja algorytmu TEA

Wstęp

Algorytmy metaheurystyczne to ogólne algorytmy pozwalające rozwiązywać różnego rodzaju problemy obliczeniowe, zwykle optymalizacyjne. Definicja tych algorytmów nie jest ścisła, natomiast mają one pewne wspólne cechy. Zazwyczaj przetwarzają pewien zbiór tymczasowych rozwiązań, które zmieniają się w trakcie działania algorytmu, podążając w coraz bardziej obiecujące obszary przestrzeni rozwiązań. Kolejna iteracja algorytmu jest budowana na podstawie rezultatów poprzedniej iteracji, a kierunek przeszukiwania jest wyznaczony przez zdefiniowaną funkcję oceny rozwiązań. Na ogół najpierw przeszukiwany jest stosunkowo duży fragment przestrzeni (eksploracja), a wraz z kolejnymi iteracjami istniejące już rozwiązania są ulepszane w mniejszych, lokalnych podobszarach (eksploatacja). Algorytmy metaheurystyczne nie gwarantują znalezienia rozwiązania optymalnego, ale często znajdują rozwiązania bliskie optymalnym w stosunkowo krótkim czasie. Są one odporne na zakłócenia i mają zdolność wychodzenia z optimów lokalnych. W niniejszej rozprawie opracowano nowe wersje wybranych algorytmów metaheurystycznych: algorytmu przeszukiwania z tabu oraz algorytmów optymalizacji mrowiskowej. Przystosowane zostały one do rozwiązania problemu kryptoanalizy szyfrów strumieniowych, które są ważnym elementem ochrony danych przetwarzanych i przechowywanych w systemach informatycznych.

Próby ukrycia informacji przed niepowołanymi osobami mają długą historię. Już w V w. p.n.e. wśród hebrajskich uczonych pojawił się prosty szyfr podstawieniowy Atbash. Choć od tego czasu wiele się zmieniło, a sztuka utajniania informacji przeszła sporą transformację, zagadnienie poufności komunikacji jest cały czas istotne. Szyfrowanie występuje często, nawet jeśli ludzie nie zdają sobie z tego sprawy. Jest ono stosowane wszędzie tam, gdzie ochrona przesyłanych lub przechowywanych danych, szczególnie w systemach informatycznych, ma kluczowe znaczenie. Z szyfrowania korzysta się podczas rozmawiania przez telefon, logowania się przez Internet do konta bankowego czy podczas dokonywania zakupów internetowych. Szyfrowanie oraz utajnianie wiadomości, a także kryptoanaliza, ma również ogromne znaczenie w wojskowości. Szyfrowanie to zagadnienie o długiej historii, nadal ważne i aktualne.

Szyfrowanie jest w systemach informatycznych jedną z najbardziej skutecznych metod zabezpieczania poufności danych, zarówno tych przechowywanych (np. w pamięciach masowych), jak i przekazywanych (przykładowo drogą telefoniczną lub za pomocą poczty elektronicznej). Tworzeniem algorytmów szyfrujących i deszyfrujących zajmuje się kryptografia. Nie wszystkie algorytmy kryptograficzne są tak samo bezpieczne i nie zawsze gwarantują pełne bezpieczeństwo danych przekazywanych i przechowywanych w systemach informatycznych. Badaniem poziomu tego bezpieczeństwa, a tym samym ujawnianiem słabości stosowanych rozwiązań kryptograficznych, zajmuje się kryptoanaliza. Kryptografia i kryptoanaliza łącznie tworzą dziedzinę wiedzy zwaną kryptologią.

Zasada działania szyfru strumieniowego, czyli takiego, jak szyfry rozpatrywane w niniejszej rozprawie, polega na osobnym szyfrowaniu każdego znaku tekstu jawnego. Metaheurystyki najczęściej dają rozwiązania przybliżone. W wielu zastosowaniach szybkie otrzymanie rozwiązania bliskiego optimum może być wystarczające, natomiast w kryptologii wymagane jest odnalezienie jednego konkretnego klucza (czyli optimum), bo każdy inny klucz, nawet bardzo bliski optimum, z założenia nie pozwoli na odszyfrowanie nawet fragmentu wiadomości. Często też krajobraz poszukiwań wyznaczony na przestrzeni możliwych rozwiązań przez funkcję oceny jest w kryptografii mało przydatny. W przestrzeni rozwiązań istnieje tylko jedno rozwiązanie o bardzo dużej wartości funkcji oceny (poszukiwany klucz), a pozostałe rozwiązania są do siebie podobne pod względem wartości funkcji oceny, która jest mała. Choć intuicyjnie algorytmy metaheurystyczne mogą nie wydawać się odpowiednie do zastosowania w kryptografii, to jednak uzyskane wyniki pokazują, że kryptoanaliza przy ich użyciu może dawać zadowalające rezultaty.

Celem niniejszej rozprawy jest zastosowanie algorytmów metaheurystycznych do kryptoanalizy szyfrów strumieniowych. W przedstawionych w rozprawie eksperymentach użyto algorytmu przeszukiwania z tabu oraz wybranych algorytmów optymalizacji mrowiskowej, które przystosowano do rozwiązania problemu kryptoanalizy. Dzięki zastosowaniu algorytmów metaheurystycznych można zdobyć informacje o stanie wewnętrznym algorytmu szyfrującego. W kryptografii odkrycie jakiegokolwiek fragmentu wiadomości lub klucza, choćby nawet jednego tylko bitu, jest już uznawane za naruszenie bezpieczeństwa i słabość danego szyfru lub jego implementacji. Zatem nawet tylko częściowe odszyfrowanie szyfrogramu lub możliwość przewidzenia tylko niektórych fragmentów strumienia szyfrującego należy traktować jako zagrożenie mogące prowadzić do pełnego złamania szyfru.

Teza rozprawy

Algorytmy metaheurystyczne stanowią skuteczne narzędzie kryptoanalityczne do łamania współczesnych szyfrów strumieniowych.

Cel rozprawy

Głównym celem rozprawy było opracowanie modyfikacji algorytmu przeszukiwania z tabu oraz wybranych algorytmów optymalizacji mrowiskowej na potrzeby kryptoanalizy szyfrów strumieniowych. Badania opisane w rozprawie koncentrują się na odnalezieniu stanu wewnętrznego algorytmu szyfrującego, mając dany fragment szyfrogramu oraz odpowiadający mu fragment tekstu jawnego. W praktyce znajomość stanu wewnętrznego pozwala uzyskać nie tylko znany fragment szyfrogramu, ale również pozostałą część strumienia szyfrującego, a co za tym idzie, odszyfrować resztę szyfrogramu.

Dodatkowe cele rozprawy to:

- analiza algorytmów metaheurystycznych pod kątem możliwości ich zastosowania w kryptoanalizie,
- analiza istniejących metod kryptoanalizy wybranych szyfrów strumieniowych,
- zaimplementowanie wybranych szyfrów strumieniowych oraz stworzenie bazy ich wektorów testowych,
- zaprojektowanie i zaimplementowanie opracowanych wersji algorytmów metaheurystycznych, umożliwiające automatyczną kryptoanalizę zadanego strumienia szyfrującego,
- przeprowadzenie badań eksperymentalnych zaproponowanych algorytmów pod kątem wpływu wartości ich parametrów na jakość uzyskiwanych rozwiązań,
- wykonanie eksperymentów na mniejszych wersjach szyfrów w celu wybrania najbardziej skutecznego algorytmu do dalszych badań,
- porównanie wyników otrzymanych przy użyciu algorytmu przeszukiwania z tabu z innymi algorytmami metaheurystycznymi pod kątem kryptoanalizy oraz z istniejącymi atakami na wybrane szyfry strumieniowe.

Częściowe wyniki przedstawione w rozprawie zostały zaprezentowane podczas międzynarodowych konferencji [Pol17, Pol18] oraz opublikowane w pracy [PB18].

Układ rozprawy

Rozprawa składa się z sześciu rozdziałów. W rozdziale 1 przedstawiono zagadnienia związane z algorytmami metaheurystycznymi, ich cechy oraz podział. Szczegółowo opisano klasyczne wersje algorytmu przeszukiwania z tabu oraz algorytmu optymalizacji

mrowiskowej, a także możliwe modyfikacje i zastosowania tych algorytmów. W rozdziale 2 wprowadzono podstawowe pojęcia kryptologii, opisano kryptoanalizę oraz szczegółowo omówiono działanie szyfrów poddanych późniejszej analizie. Przedstawiono również dokładny opis szyfrowania strumieniowego. W rozdziale 3 zawarto przegląd algorytmów metaheurystycznych stosowanych w kryptoanalizie. Przeglądu dokonano z podziałem na szyfry klasyczne, blokowe, strumieniowe oraz asymetryczne. Przegląd uzupełniono zastosowaniami algorytmów metaheurystycznych w kryptografii. W rozdziale 4 zawarto szczegółowy opis autorskiej wersji algorytmu kryptoanalizy opartego na przeszukiwaniu z tabu, wraz z opisem wartości parametrów użytych w badaniach eksperymentalnych. Przedstawiono także prosty przykład działania algorytmu. W rozdziale 5 zawarto szczegółowy opis autorskich wersji algorytmów kryptoanalizy opartych na algorytmie optymalizacji mrowiskowej, wraz z opisem wartości parametrów użytych w badaniach. Rozdział 6 przedstawia opis badań eksperymentalnych z zastosowaniem opracowanych wersji algorytmów w kryptoanalizie wybranych szyfrów strumieniowych: RC4, VMPC, RC4+. Ten rozdział zawiera szczegółowy opis warunków przeprowadzonych eksperymentów oraz użyte wartości parametrów. Wyniki badań są przedstawione osobno dla każdego analizowanego szyfru strumieniowego. Otrzymane wyniki są także porównane z innymi znanymi z literatury atakami, próbami wygenerowania stanu wewnętrznego w sposób losowy oraz losowo generowanymi ciągami liczb o określonej długości. W ostatnim rozdziale zawarto podsumowanie rozprawy oraz zaproponowano dalsze kierunki badań.

Rozdział 1

Algorytmy metaheurystyczne

Metaheurystyki to takie strategie przeszukiwania przestrzeni rozwiązań, które stanowią pewien uniwersalny schemat przybliżonego rozwiązywania problemów, głównie optymalizacyjnych. U podłoża wielu metaheurystyk leżą zjawiska występujące w naturze, dlatego duża część z nich nazywana jest algorytmami inspirowanymi naturą. Dzięki temu, podobnie jak zachowania naturalne, takie algorytmy łatwo dostosowują się do analizowanego problemu, a niekiedy i warunków zmiennych w czasie. Charakterystyczne jest również to, że sposób działania jest prosty, choć używany do rozwiązywania złożonych problemów. Metaheurystyki mogą być również nazywane technikami optymalizacyjnymi. Istotną właściwością metaheurystyk jest zdolność wybicia się z optimum lokalnego. Stosowane są dla problemów, dla których algorytmy dokładne nie istnieją lub są zbyt kosztowne pod względem czasu obliczeniowego [Glo86, Mic06].

1.1 Wprowadzenie

Definicja metaheurystyki nie jest ścisła; na to pojęcie składa się wiele różnych algorytmów, które współdzielą pewne wspólne cechy [BR03, GK03]:

- mają zastosowanie w problemach, dla których możliwe jest zbudowanie przestrzeni rozwiązań,
- analizują pewną klasę rozwiązań, które zmieniają się w czasie,
- definiują funkcję modyfikującą rozwiązanie,
- rozwojem steruje zdefiniowana funkcja oceny lub kosztu (w zależności od potrzeb funkcja ta może być maksymalizowana lub minimalizowana),

- są to algorytmy iteracyjne, kolejna iteracja jest budowana na podstawie poprzedniej,
- na kolejnych etapach przechodzi się od eksploracji do eksploatacji,
- są to algorytmy niedeterministyczne,
- nie gwarantują znalezienia rozwiązania optymalnego, ale najczęściej znajdują rozwiązanie bliskie optymalnemu w stosunkowo krótkim czasie (w porównaniu z algorytmami dokładnymi).

W porównaniu do algorytmów klasycznych algorytmy metaheurystyczne mają więcej parametrów, którymi trzeba sterować. Często od właściwego doboru wartości tych parametrów zależy sukces lub porażka algorytmu w zastosowaniu do danego problemu. Z drugiej strony, algorytmy metaheurystyczne znacznie łatwiej adaptują się do aktualnej sytuacji – zmiana modelu problemu lub funkcji oceny najczęściej spowoduje tylko krótkotrwałe zakłócenia. Dodatkowo, są one odporne na szum i zakłócenia, więc dobrze nadają się do rozwiązywania większości rzeczywistych problemów. Przykładowe algorytmy metaheurystyczne zostały zebrane w tab. 1.1. Zestawienie to w kolejnych kolumnach zawiera następujące informacje:

1. rok wprowadzenia danego algorytmu metaheurystycznego,
2. jego autora lub autorów,
3. polską nazwę,
4. nazwę w języku angielskim (najczęściej będzie to nazwa oryginalna),
5. skrót (będący ogólnie przyjętym akronimem, pochodzącym od nazwy w języku angielskim),
6. naśladowane zjawisko.

W zależności od tego, jakimi cechami się charakteryzują, algorytmy metaheurystyczne można podzielić na grupy:

- algorytmy analizujące w danej iteracji tylko jedno rozwiązanie (np. symulowane wyżarzanie [Čer85, KGV83], przeszukiwanie z tabu [Glo86]) lub algorytmy utrzymujące wiele rozwiązań jednocześnie (np. algorytm genetyczny [Hol75], optymalizacja mrowiskowa [DDC99], optymalizacja stadna cząsteczek [KE95]),
- algorytmy inspirowane zjawiskami fizycznymi, np. symulowane wyżarzanie [Čer85, KGV83],

Tabela 1.1: Spis algorytmów metaheurystycznych wykorzystywanych w kryptoanalizie

Rok	Autor(zy)	Nazwa polska	Nazwa angielska	Skrót	Naśladowane zjawisko
1943	W. S. McCulloch, W. Pitts [MP43]	sztuczna sieć neuronowa	*Artificial Neural Network*	ANN	budowa neuronów, synaps oraz całych układów nerwowych, przede wszystkim budowa i działanie mózgu
1951- -1966	J. von Neumann [vN51, vN66]	automat komórkowy	*Cellular Automaton*	CA	samoreprodukcja organizmów żywych
lata 70. XX w.	–	metoda wspinaczkowa	*Hill Climbing*	HC	poprawianie jakości rozwiązania poprzez niewielkie modyfikacje
1975	J. Holland [Hol75]	algorytm genetyczny	*Genetic Algorithm*	GA	darwinowska teoria ewolucji organizmów żywych, wraz z selekcją, krzyżowaniem i mutacją
1983 1985	S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi [KGV83] V. Černý [Čer85]	symulowane wyżarzanie	*Simulated Annealing*	SA	proces wyżarzania metali w metalurgii
1986	F. Glover [Glo86]	przeszukiwanie z tabu	*Tabu Search*	TS	pamięć poprzednich ruchów i uczenie się na ich podstawie
1989	P. Moscato [Mos89]	algorytm memetyczny	*Memetic Algorithm*	MA	ewolucja memów, czyli jednostek informacji kulturowej
1991	M. Dorigo [DMC91, Dor92]	system mrówkowy	*Ant System*	AS	odnajdywanie przez mrówki pożywienia, w tym zjawisko stygmergii, czyli porozumiewanie się przez ślad feromonowy
1999	M. Dorigo, G. Di Caro, L. M. Gambardella [DDC99]	algorytm optymalizacji mrowiskowej	*Ant Colony Optimization*	ACO	
1994	M. Srinivas, L. M. Patnaik [SP94]	adaptacyjny algorytm genetyczny	*Adaptive Genetic Algorithm*	AGA	darwinowska teoria ewolucji organizmów żywych, wraz z selekcją, krzyżowaniem i mutacją
1995	J. Kennedy, R. Eberhart [KE95]	optymalizacja stadna cząsteczek	*Particle Swarm Optimization*	PSO	społeczne poszukiwanie pożywienia oraz poruszanie się stad zwierząt i ptaków oraz ławic ryb
1997	T. M. Mitchell [Mit97]	uczenie maszynowe	*Machine Learning*	ML	uczenie się i wyciąganie nowych wniosków na podstawie poprzednich zdarzeń
1997	R. Storn, K. Price [SP97]	ewolucja różnicowa	*Differential Evolution*	DE	darwinowska teoria ewolucji organizmów żywych, wraz z selekcją, krzyżowaniem i mutacją
2002	K. M. Passino [Pas02]	optymalizacja żerowania bakterii	*Bacterial Foraging Optimization*	BFO	społeczne zachowanie bakterii E.coli w czasie żerowania oraz zjawisko chemotaksji
2005	D. T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, M. Zaidi [POR+05]	algorytm pszczeli	*Bees Algorithm*	BA	zachowanie pszczół miodnych w czasie żerowania
2008	X.-S. Yang [Yan08]	algorytm świetlika	*Firefly Algorithm*	FA	sposób komunikacji świetlików przez rozbłyski świetlne
2009	X.-S. Yang, S. Deb [YD09]	system kukułczy	*Cuckoo Search*	CS	pasożytnictwo lęgowe kukułek (podrzucanie przez kukułki swoich jaj do gniazd innych ptaków) oraz loty Lévy'ego

- algorytmy inspirowane zachowaniami zwierząt, np. system kukułczy [YD09], algorytm świetlika [Yan08], algorytm pszczeli [POR+05],
- algorytmy wykorzystujące inteligencję rozproszoną, inteligencję roju, np. optymalizacja mrowiskowa [DDC99], optymalizacja stadna cząsteczek [KE95], algorytm pszczeli [POR+05],
- algorytmy ewolucyjne, np. algorytm genetyczny [Hol75], ewolucja różnicowa [SP97],
- algorytmy wspinania się, np. symulowane wyżarzanie [Čer85, KGV83], przeszukiwanie z tabu [Glo86].

Ponadto możliwa jest hybrydyzacja algorytmów metaheurystycznych zarówno z algorytmami klasycznymi, jak i innymi metaheurystykami.

W związku z wykorzystaniem w rozprawie przeszukiwania z tabu oraz algorytmu optymalizacji mrowiskowej zostaną one opisane w dalszej części rozdziału.

1.2 Przeszukiwanie z tabu

Przeszukiwanie z tabu (ang. *tabu search*, TS) zostało wprowadzone przez F. Glovera w 1986 roku [Glo86]. W polskiej literaturze można się również spotkać z nazwami: „przeszukiwanie z zakazami" lub „poszukiwanie z tabu/zakazami". Główną ideą przeszukiwania z tabu jest bazowanie na poprzednich doświadczeniach, by podejmować lepsze decyzje w kolejnych krokach. Istotnym elementem tej techniki jest pamięć poprzednich ruchów. Ruchem będzie przejście z bieżącego rozwiązania do najlepszego rozwiązania z otoczenia. Pamięć poprzednio wykonanych ruchów niejako ukierunkowuje przeszukiwanie przestrzeni rozwiązań. Pamięć ta jest reprezentowana w formie listy tabu (stąd nazwa) ostatnio wykonanych ruchów, które jednocześnie są ruchami zabronionymi w najbliższej przyszłości. Okres, przez który dany ruch będzie znajdował się na liście tabu, nazywany jest horyzontem (lub kadencją) i jest to określona liczba iteracji.

1.2.1 Wersja podstawowa

Klasyczną wersję przeszukiwania z tabu przedstawiono jako alg. 1.1, gdzie:

TABU – lista tabu,
k – licznik pętli,
I – maksymalna liczba iteracji,
$R(r)$ – otoczenie punktu r,
$rBest$ – najlepsze rozwiązanie znalezione do tej pory,

f_{fit} – funkcja dopasowania.

W przeciwieństwie do prostego poszukiwania lokalnego, technika ta umożliwia unikanie lokalnego optimum w ramach jednego uruchomienia. Jest to sposób oparty na pamięci (liście tabu), która zmusza algorytm do badania nowych obszarów przestrzeni przeszukiwania. W kolejnej iteracji nie musi być odnaleziony lepszy punkt z otoczenia. Wystarczy, że będzie to zaakceptowane rozwiązanie z otoczenia rozwiązania aktualnie rozpatrywanego. Akceptacja ta jest oparta na historii poszukiwania. Jako kolejne bieżące rozwiązanie wybierane jest takie, które nie znajduje się na liście tabu, bez względu na to, czy nowe rozwiązanie ma większą wartość funkcji dopasowania od bieżącego rozwiązania czy też nie. To często pozwala na wybicie się z optimum lokalnego.

Algorytm 1.1: PROSTE PRZESZUKIWANIE Z TABU

Wejście: I
Wyjście: najlepsze znalezione rozwiązanie

```
1  wybierz początkowe r
2  rBest ← r
3  TABU ← {}
4  l ← 0
5  while (l < I) && (R(r) − TABU ≠ ∅) do
6      r = arg min_{i∈R(r)−TABU} (f_fit(i))
7      if f_fit(r) < f_fit(rBest) then
8          rBest ← r
9      uaktualnij TABU
10     l ← l + 1
11 return rBest
```

Otoczeniem (sąsiedztwem) jest zbiór rozwiązań podobnych do rozwiązania rozpatrywanego i różniących się od niego tylko w nieznacznym stopniu. Dobrze zdefiniowane otoczenie zawiera co najmniej jedno rozwiązanie i nie obejmuje całej przestrzeni rozwiązań. Niezależnie od wyboru rozwiązania początkowego każde rozwiązanie w całej przestrzeni możliwych rozwiązań powinno być osiągalne przy użyciu relacji sąsiedztwa.

Dobór właściwej wielkości horyzontu w klasycznej wersji przeszukiwania z tabu jest dość trudny. Krótszy horyzont zwiększa dokładność przeszukiwania (intensyfikacja), z drugiej jednak strony zwiększa ryzyko wpadnięcia w cykl w okolicach lokalnego optimum. Z kolei większy horyzont to większy zakres przeszukiwania (dywersyfikacja). Jednak w takim przypadku istnieje ryzyko pogorszenia jakości uzyskanych wyników, gdyż otoczenie dobrych rozwiązań nie jest dokładnie przeszukiwane. Rozmiar horyzontu jest zazwyczaj powiązany pewną funkcją z rozmiarem otoczenia lub rozmiarem problemu,

dzięki czemu wraz ze zmniejszeniem lub zwiększeniem rozmiaru problemu możliwe jest automatyczne dostosowanie długości listy tabu, czyli horyzontu.

Przeszukiwanie z tabu to iteracyjne przeszukiwanie otoczenia w przestrzeni rozwiązań. Aby jednak nie trwało to w nieskończoność, nakłada się pewne ograniczenie co do długości trwania wykonania algorytmu. Ograniczenie to jest nazywane warunkiem zatrzymania lub warunkiem stopu i może być zdefiniowane w różny sposób, w zależności od potrzeb. Najczęściej spotykane warunki zatrzymania to:

- z góry określona liczba iteracji lub określony czas obliczeń,
- pewna liczba kolejnych iteracji, w trakcie których nie nastąpiła poprawa jakości rozwiązania,
- znalezienie rozwiązania o określonej z góry jakości (zdefiniowanej najczęściej jako konkretna wartość funkcji dopasowania).

Przez 30 lat od powstania przeszukiwania z tabu powstało wiele różnych jego wersji i usprawnień. Kilka z nich zostanie przedstawionych w niniejszym podrozdziale. Podobnie jak z samymi algorytmami metaheurystycznymi, tak i poszczególne modyfikacje mogą prowadzić do lepszych rezultatów w przypadku jednych problemów, a inne modyfikacje w przypadku innych. Nie każda modyfikacja będzie pozwalała na poprawienie wyników problemów każdego rodzaju. Każdy przypadek należy rozpatrywać indywidualnie i rozważyć, na ile dany algorytm wraz z modyfikacją ma szansę dać dobre efekty, a następnie przeprowadzić serię eksperymentów, by potwierdzić lub obalić swoją tezę.

Modyfikacje przeszukiwania z tabu można podzielić na trzy główne kategorie. Są to: modyfikacje procedury wyboru rozwiązania do kolejnej iteracji, modyfikacje wyboru otoczenia oraz modyfikacje dotyczące horyzontu. Przykładowe modyfikacje każdego typu zostaną opisane w poniższych punktach. Wpływ wybranych modyfikacji na jakość otrzymywanych rozwiązań dla problemu kryptoanalizy zostanie również przetestowany w rozdz. 6.

1.2.2 Modyfikacje procedury selekcji

Poniżej przedstawiono modyfikacje dotyczące procedury selekcji aktualnego rozwiązania do kolejnej iteracji algorytmu.

Kryterium aspiracji

Prawdopodobnie najbardziej znaną modyfikacją TS jest kryterium aspiracji [GK03]. Może się zdarzyć tak, że ruch prowadzący do bardzo dobrego rozwiązania znajdzie się na liście tabu. W takim przypadku można pominąć fakt, że ten ruch jest zabroniony

i wybrać go do następnej iteracji. To właśnie jest kryterium aspiracji, które anuluje zakaz wykonania danego ruchu wynikający z jego obecności na liście tabu. Gdy zabroniony ruch jest wystarczająco dobry, można go mimo wszystko wykonać.

Samo kryterium aspiracji również może być różnie zdefiniowane. W najprostszej postaci będzie to akceptacja ruchu, który prowadzi do rozwiązania najlepszego w całym wykonaniu algorytmu. Innymi słowy proponowane rozwiązanie jest lepsze niż najlepsze do tej pory znalezione (wartość jego funkcji dopasowania jest wyższa niż najlepszego znalezionego dotąd rozwiązania).

Mniej restrykcyjna wersja zakłada możliwość akceptacji zabronionego ruchu, gdy prowadzi on do rozwiązania lepszego od aktualnie rozpatrywanego o jakąś ustaloną z góry wartość funkcji dopasowania. Wielkość tego parametru powinna być każdorazowo ustalona eksperymentalnie.

Uwzględnianie kryterium aspiracji wymaga analizy całego otoczenia, bez względu na to, które ruchy znajdują się aktualnie na liście tabu. Bez wyznaczenia jakości wszystkich rozwiązań z otoczenia niemożliwe będzie odnalezienie potencjalnych rozwiązań, które mogłyby zostać zakwalifikowane do kryterium aspiracji.

Probabilistyczna procedura selekcji

Kolejna modyfikacja jest również związana z wyborem rozwiązania do kolejnej iteracji. Klasyczna wersja przeszukiwania z tabu w wyborze rozwiązania do kolejnej iteracji jest deterministyczna – zawsze wybierane jest rozwiązanie o najlepszym dopasowaniu. W wersji probabilistycznej lepsze dopasowanie to tylko większa szansa na wybór do kolejnej iteracji, wybór jednak nie jest deterministyczny. Lepiej dopasowane rozwiązania mają proporcjonalnie większe szanse na przejście do kolejnej iteracji, ale sam wybór rozwiązania jest dokonywany w sposób losowy. Podejście takie wpływa na zwiększenie różnorodności wyników.

1.2.3 Modyfikacje wyboru otoczenia

W przypadku, gdy otoczenie rozpatrywanego rozwiązania jest bardzo duże, możliwe jest zmniejszenie złożoności obliczeniowej przez analizę tylko pewnego podzbioru otoczenia. Najczęściej rozwiązania do analizy są wybierane losowo spośród całego otoczenia. Wielkość analizowanego zbioru jest zależna od problemu i należy ją indywidualnie dobrać do zadanego przypadku.

Ponieważ nie są analizowane wszystkie rozwiązania z otoczenia, możliwe jest pominięcie optimum globalnego. Aby zminimalizować szanse wystąpienia takiego przypadku, warto co kilka lub kilkanaście iteracji zbadać dokładnie całe otoczenie. Nie

powinno to znacznie wpłynąć na złożoność obliczeniową, a istotnie zwiększa szanse odnalezienia optymalnego rozwiązania.

1.2.4 Modyfikacje horyzontu

Inna zmiana klasycznego przeszukiwania z tabu to modyfikacja sposobu określania horyzontu dla poszczególnych ruchów.

Losowy horyzont

Zamiast jednej wartości stałej dla wszystkich zabronionych ruchów, można każdorazowo losować wartość horyzontu z ustalonego z góry przedziału. W związku z tym długość listy tabu będzie się wahać w poszczególnych iteracjach. W takim przypadku każdy ruch będzie znajdował się na liście tabu przez różną liczbę iteracji.

Horyzont adaptacyjny

Horyzont można również modyfikować w zależności od przebiegu algorytmu. Gdy wartość funkcji dopasowania zwiększa się z każdą kolejną iteracją, zmniejszenie horyzontu pozwoli na większą eksploatację dobrze rokującego obszaru przestrzeni rozwiązań. Z kolei gdy w kolejnych iteracjach wartość funkcji dopasowania ulega stagnacji lub wręcz pogorszeniu, warto zwiększyć horyzont, by pobudzić eksplorację. Podobnie jak w przypadku horyzontu losowego, rozmiar tak dobieranego horyzontu powinien być ograniczony pewnymi wartościami brzegowymi.

Horyzont wybierany cyklicznie

Wielkość horyzontu może być wybierana cyklicznie z pewnego z góry zadanego ciągu wartości. Na przykład dla ciągu (2, 17, 3, 13, 5, 11, 7) dla ruchu wybranego w pierwszej iteracji horyzont będzie miał wartość 2, dla ruchu z drugiej iteracji – 17, w kolejnej iteracji – 3 itd. Po wybraniu horyzontu równego 7, kolejny ruch zostanie zakazany ponownie na 2 iteracje i tak cyklicznie. Sposób wyboru wartości i długości ciągu wielkości horyzontów dla każdego problemu powinien być ustalony eksperymentalnie.

Pamięć częstości użycia

Przeszukiwanie z tabu można wzbogacić o pamięć długotrwałą w postaci pamięci częstości wykonania danych ruchów. Oprócz listy zakazanych ruchów z ostatnich iteracji (pamięć krótkotrwała) można dodać dodatkową strukturę, która będzie przechowywać informację, ile razy każdy ruch został wykonany w przeszłych iteracjach. Ta struktura jest nazywana pamięcią częstości użycia. Jej horyzont jest znacznie większy niż listy tabu i może

być określony liczbowo (ile ostatnich iteracji jest branych pod uwagę) lub może zawierać wszystkie użycia od początku uruchomienia algorytmu.

Pamięć częstości użycia pozwala na uwzględnienie wiedzy dotyczącej frekwencji wykonywanych ruchów dzięki spojrzeniu na znacznie szerszy horyzont niż horyzont podstawowej listy tabu. Pamięć długotrwała w tej postaci pozwala różnicować przeszukiwanie przestrzeni możliwych rozwiązań. Korzystając z informacji o tym, które ruchy pojawiały się rzadko lub nie pojawiały się wcale, można zwiększyć eksplorację poprzez sprawdzenie tych właśnie możliwości.

1.2.5 Zastosowania przeszukiwania z tabu

Pomimo swojej prostoty przeszukiwanie z tabu przynosi bardzo dobre wyniki tam, gdzie dokładne algorytmy nie są znane lub są nierealizowalne w praktyce. Z tego powodu aktualne zastosowania TS obejmują wiele dziedzin i zagadnień. Są to między innymi:

- harmonogramy dla pracowników [GM86],
- kolorowanie grafu [HdW87],
- telekomunikacja [LG93],
- problem maksymalnej kliki [GSS93],
- problem marszrutyzacji [BO99],
- klasyfikacja wzorców [ZS02],
- gospodarka odpadami [KD08],
- dystrybucja energii [WW11],
- planowanie zasobów [MS17],
- bioinformatyka [RHKB17].

Jednak, jak dotąd, nie było badań na temat tego podejścia w kryptoanalizie szyfrów strumieniowych.

1.3 Algorytm optymalizacji mrowiskowej

Algorytmy optymalizacji mrowiskowej (ang. *ant colony optimization*, ACO) stanowią całą grupę algorytmów, których działanie opiera się na zjawiskach komunikacji pozawerbalnej w koloniach mrówek. Jako pierwszy został wprowadzony system

mrówkowy (ang. *ant system*, AS) przez M. Dorigo [DMC91, Dor92] w 1991 roku do rozwiązania problemu komiwojażera. Inspiracją do stworzenia tego algorytmu był sposób porozumiewania się mrówek w środowisku naturalnym. Stygmergia, czyli pośredni sposób komunikacji, pozwala społeczności mrówek znajdować najkrótsze ścieżki do źródeł pożywienia oraz omijać przeszkody. Co więcej, sposób ten pozwala również reagować na środowisko zmienne w czasie, przykładowo na wyczerpanie źródła pożywienia lub pojawienie się nowej przeszkody. Stygmergia w koloniach mrówek jest możliwa dzięki substancji chemicznej nazwanej feromonem.

1.3.1 Ogólna struktura algorytmów optymalizacji mrowiskowej

Główną ideą algorytmów optymalizacji mrowiskowej jest porozumiewanie się pozawerbalne pomiędzy agentami-mrówkami poprzez zmiany dokonywane we wspólnym środowisku w celu znalezienia optymalnego rozwiązania. Jest to algorytm wieloagentowy, którego działanie jest niedeterministyczne. Klasyczny algorytm optymalizacji mrowiskowej, rozwiązujący problem komiwojażera, został przedstawiony jako alg. 1.2.

Algorytm 1.2: ALGORYTM OPTYMALIZACJI MROWISKOWEJ

Wejście: macierz odległości miast, liczba mrówek, liczba iteracji
Wyjście: najkrótsza znaleziona trasa

```
1  inicjuj
2  forall cykl do
3      rozmieść mrówki
4      forall iteracja do
5          forall mrówka do
6              wyznacz krawędź              {Wybór krawędzi zgodnie z regułą przejścia}
7              dodaj krawędź do rozwiązania
8              aktualizuj feromon lokalnie {Krok opcjonalny}
9      odparuj feromon                      {Krok opcjonalny}
10     aktualizuj feromon globalnie         {Krok opcjonalny}
11 return najlepsze znalezione rozwiązanie (najkrótsza trasa)
```

Wybór kolejnego wierzchołka (miasta) jest przez mrówkę dokonywany w sposób probabilistyczny. Każda mrówka posiada pewien rodzaj pamięci, w której przechowuje zbiór odwiedzonych miast (lista $tabu$). W trakcie budowania rozwiązania na kolejny ruch mrówki mają wpływ:

- ślad feromonowy,
- informacja heurystyczna charakterystyczna dla danego problemu.

Ślad feromonowy jest istotnym elementem tego algorytmu. Jest on zarówno odkładany, jak i odczytywany przez poszczególnych agentów-mrówki. Ów ślad ulega

również naturalnemu wyparowaniu. Sposób zarówno odkładania, jak i odczytywania oraz wyparowywania feromonu jest kwestią parametryzacji algorytmu.

Informacja heurystyczna pochodzi z danego problemu. Przykładowo, dla problemu komiwojażera będzie to długość krawędzi dostępnych do wyboru (czyli prowadzących do wierzchołków nie znajdujących się na liście $tabu$ danej mrówki). Atrakcyjne są zarówno te wierzchołki (miasta), do których prowadzą najkrótsze krawędzie, jak i te, na których jest duża wartość śladu feromonowego. Odpowiednie zrównoważenie wpływu śladu feromonowego oraz informacji heurystycznej na probabilistyczny wybór kolejnego kroku przez mrówkę jest podstawą sukcesu osiąganego przez algorytmy optymalizacji mrowiskowej.

Jeśli jedynie ślad feromonowy będzie uwzględniony, szybko doprowadzi to do stagnacji w rejonie suboptymalnym. Oznacza to, że wszystkie mrówki poruszają się tą samą ścieżką i budują te same rozwiązania. Jeśli uwzględniona będzie tylko informacja heurystyczna, algorytm niewiele się będzie różnił od algorytmu zachłannego.

Wyparowywanie feromonu to przydatna forma „zapominania", promująca eksplorację nowych rejonów przestrzeni rozwiązań. Wartość feromonu zmniejsza się wraz z upływem czasu, co jest potrzebne do uniknięcia zbyt szybkiej zbieżności do optimum lokalnego. Wyparowywanie feromonu pozwala uniknąć nieograniczonej kumulacji feromonu, dzięki czemu algorytm optymalizacji mrowiskowej jest w stanie znaleźć dobre rozwiązanie, pomimo podjęcia złych decyzji w trakcie jego trwania. Na krawędziach, które nie są wybierane przez mrówki, wartość feromonu będzie malała wykładniczo [DS03].

Warto zwrócić uwagę, że poszczególne mrówki poruszają się jednocześnie oraz niezależnie od siebie. Każda z nich jest w stanie samodzielnie zbudować jakieś rozwiązanie rozpatrywanego problemu, choć jego jakość raczej nie będzie zadowalająca. Zazwyczaj, dobrej jakości rozwiązania powstają wskutek kolektywnej interakcji pomiędzy mrówkami. Interakcja ta jest osiągana poprzez ślad feromonowy obecny we wspólnym środowisku.

Możliwe jest działanie algorytmu optymalizacji mrowiskowej z populacją wielkości jednej mrówki. Niemniej jednak dowody eksperymentalne pokazują, że lepsze rezultaty uzyskiwane są, gdy liczba mrówek jest większa niż 1. Należy jednak wziąć pod uwagę, że zbyt duża liczba mrówek może wpłynąć negatywnie na efektywność rozwiązywania problemu. Najlepiej dobrać liczbę mrówek zależną od klasy rozwiązywanego problemu oraz jako funkcję rozmiaru tego problemu. Najczęściej optymalny dobór liczby mrówek wymaga testów eksperymentalnych. Na szczęście algorytm optymalizacji mrowiskowej jest w miarę nieczuły na liczbę mrówek [DS03].

Pojedyncze mrówki jako takie nie mają zdolności adaptacyjnej, jednakże adaptacyjnie zmieniają sposób postrzegania problemu przez inne mrówki. Pojedyncza mrówka – zarówno naturalna, jak i sztuczna – nie posiada inteligencji lub posiada inteligencję bardzo znikomą.

Dobrej jakości rozwiązania powstają jako efekt inteligencji zbiorowej kolonii mrówek. Sztuczne mrówki, w przeciwieństwie do swoich naturalnych pierwowzorów, są dodatkowo wyposażone w pamięć, która pozwala zaimplementować ograniczenia nakładane przez problem, a także przechowywać rozwiązania częściowe w trakcie ich budowy. Posiadają również zdolność deponowania ilości feromonu proporcjonalnej do jakości produkowanych rozwiązań.

Ma tutaj miejsce proces autokatalizy. Oznacza to, że decyzja podjęta w czasie t zwiększa prawdopodobieństwo podjęcia tej samej decyzji w czasie $\mathcal{T} > t$. W algorytmie optymalizacji mrowiskowej występuje pozytywne sprzężenie zwrotne, związane z istnieniem mapy feromonowej.

Czas w algorytmie optymalizacji mrowiskowej jest symulowany w sposób dyskretny. Algorytm optymalizacji mrowiskowej, podobnie jak inne algorytmy metaheurystyczne, jest algorytmem iteracyjnym. W danej iteracji wszystkie mrówki budują swoje rozwiązania jednocześnie. Wszystkie posiadają ten sam cel, choć każda zaczyna z innego miejsca w przestrzeni rozwiązań. Również w przypadku tego algorytmu określa się pewne ograniczenia co do długości trwania. Warunki zatrzymania mogą być określone podobnie jak w przypadku przeszukiwania z tabu (podrozdz. 1.2).

1.3.2 Rodzaje algorytmów optymalizacji mrowiskowej

Pierwszym systemem mrówkowym był AS (ang. *ant system*) [DMC91]. Wszystkie pozostałe algorytmy mrówkowe i mrowiskowe są bezpośrednimi rozszerzeniami algorytmu AS w celu poprawy jego wydajności oraz jakości uzyskanych wyników.

System mrówkowy: feromon stały, średni i cykliczny

Intensywność śladu feromonowego można obliczyć z wzoru:

$$\tau_{b,c}(t+1) = (1-\rho)\tau_{b,c} + \Delta\tau_{b,c}(t, t+1), \tag{1.1}$$

gdzie:

$\tau_{b,c}(t)$ – intensywność śladu feromonowego na krawędzi pomiędzy wierzchołkami b i c w czasie t,
ρ – współczynnik wyparowywania feromonu,
$\Delta\tau_{b,c}(t, t+1)$ – ilość feromonu odłożonego na krawędzi (b, c) pomiędzy czasem t a $t+1$, określona wzorem:

$$\Delta\tau_{b,c}(t, t+1) = \sum_{k=1}^{A} \Delta\tau_{b,c}^{k}(t, t+1), \tag{1.2}$$

gdzie:

$\Delta\tau_{b,c}^{k}(t,t+1)$ – ilość feromonu odłożonego przez k-tą mrówkę na krawędzi (b,c) pomiędzy czasem t a $t+1$.

Gdy mrówka k znajduje się w wierzchołku b, to prawdopodobieństwo jej przejścia do wierzchołka c jest dane wzorem:

$$p_{b,c}(t) = \begin{cases} \dfrac{[\tau_{b,c}(t)]^{\alpha} \cdot [\eta_{b,c}]^{\beta}}{\sum\limits_{c \notin tabu_k} [\tau_{b,c}(t)]^{\alpha} \cdot [\eta_{b,c}]^{\beta}}, & \text{dla } c \notin tabu_k, \\ \\ 0, & \text{w przeciwnym razie,} \end{cases} \tag{1.3}$$

gdzie:

α – ważność śladu feromonowego,
β – ważność informacji heurystycznej,
$\tau_{b,c}(t)$ – intensywność śladu feromonowego na krawędzi pomiędzy wierzchołkami b i c w czasie t,
$\eta_{b,c}$ – informacja heurystyczna pochodząca z analizowanego problemu (widoczność miasta c z miasta b – odwrotność odległości),
$tabu_k$ – lista wierzchołków odwiedzonych przez mrówkę k.

W przypadku feromonu stałego (ang. *ant-density*) stała wartość feromonu F jest odkładana na krawędzi za każdym razem, gdy mrówka przechodzi z jednego wierzchołka do innego:

$$\Delta\tau_{b,c}^{k}(t,t+1) = \begin{cases} F, & \text{jeśli } k\text{-ta mrówka przechodzi z wierzchołka} \\ & b \text{ do } c \text{ pomiędzy czasem } t \text{ a } t+1, \\ \\ 0, & \text{w przeciwnym razie.} \end{cases} \tag{1.4}$$

W przypadku feromonu średniego (ang. *ant-quantity*) wartość odkładanego na krawędzi śladu feromonowego F jest odwrotnie proporcjonalna do jakości krawędzi $d_{b,c}$:

$$\Delta\tau_{b,c}^{k}(t,t+1) = \begin{cases} \frac{F}{d_{b,c}}, & \text{jeśli } k\text{-ta mrówka przechodzi z wierzchołka} \\ & b \text{ do } c \text{ pomiędzy czasem } t \text{ a } t+1, \\ \\ 0, & \text{w przeciwnym razie.} \end{cases} \tag{1.5}$$

W tym modelu widoczność krótszych krawędzi jest wzmocniona śladem feromonowym i są one bardziej widoczne dla mrówek. W obu przypadkach aktualizacja mapy feromonowej następuje w sposób lokalny, to jest po wykonaniu każdego kolejnego kroku przez mrówkę.

W przypadku feromonu cyklicznego (ang. *ant-cycle*) aktualizacja mapy feromonowej odbywa się dopiero po zbudowaniu rozwiązań przez wszystkie mrówki (czyli po wykonaniu e kroków). Wartość feromonu F odłożonego przez poszczególne mrówki jest określona funkcją na podstawie jakości (długości) zbudowanego rozwiązania L_k:

$$\Delta\tau_{b,c}^{k}(t,t+e) = \begin{cases} \frac{F}{L_k}, & \text{jeśli } k\text{-ta mrówka użyła w swoim cyklu krawędzi } (b,c), \\ 0, & \text{w przeciwnym razie.} \end{cases} \tag{1.6}$$

Rozwiązania uzyskane przy użyciu tego modelu są lepsze niż przy użyciu dwóch poprzednich [DMC91, Dor92].

System mrowiskowy

W systemie mrowiskowym (ang. *Ant Colony System*, ACS) [DG97], w porównaniu z algorytmem cyklicznym, występuje dodatkowe lokalne uaktualnienie śladu feromonowego po każdym kroku wykonanym przez mrówkę. Po przejściu mrówki k z wierzchołka b do wierzchołka c następuje odparowanie feromonu zgodnie z wzorem:

$$\tau_{b,c} = (1-\rho)\cdot\tau_{b,c} + \rho\cdot\tau_0, \tag{1.7}$$

gdzie:

$\tau_{b,c}$ – intensywność śladu feromonowego na krawędzi pomiędzy wierzchołkami b i c,
ρ – współczynnik wyparowania śladu feromonowego (lokalnie),
τ_0 – ślad domyślny (początkowy, inicjalny).

Efektem tego uaktualnienia jest obniżenie wartości śladu feromonowego na krawędziach odwiedzonych przez mrówkę w trakcie budowania rozwiązań. W ten sposób te krawędzie stają się mniej atrakcyjne dla innych mrówek. Mechanizm ten zmniejsza prawdopodobieństwo wybrania tej samej krawędzi przez inne mrówki, a tym samym zwiększa eksplorację przestrzeni rozwiązań.

Oprócz lokalnego uaktualniania śladu feromonowego, po zbudowaniu trasy przez wszystkie mrówki następuje również globalne uaktualnienie śladu feromonowego, zgodnie z wzorem:

$$\tau_{b,c} = (1-\sigma)\cdot\tau_{b,c} + \frac{\sigma}{L_{best}}, \tag{1.8}$$

gdzie:

σ – współczynnik wyparowania śladu feromonowego (globalnie),
L_{best} – długość najlepszej trasy w danej iteracji.

Do globalnego uaktualnienia ma prawo tylko mrówka, która w danym cyklu (iteracji) osiągnęła rozwiązanie o najlepszej jakości (najwyższej lub najniższej wartości funkcji dopasowania w zależności od rodzaju rozwiązywanego problemu). Wielkość odłożonego przez nią śladu feromonowego jest uzależniona od jakości osiągniętego rozwiązania – im lepsze, tym większy ślad feromonowy zostanie przez najlepszą mrówkę odłożony na trasie, którą przebyła. Jako najlepsza trasa może być również użyta trasa mrówki globalnie najlepszej od początku uruchomienia algorytmu [DS03].

Ponadto wprowadzono dodatkowy parametr $0 \leqslant q_0 \leqslant 1$, który steruje względną zależnością pomiędzy eksploracją a eksploatacją. Jeśli wylosowana liczba q będzie mniejsza lub równa q_0, wybór kolejnego miasta (wierzchołka) c będzie określony wzorem:

$$c = \arg \max_{a \notin tabu_k} \{[\tau_{b,a}] \cdot [\eta_{b,a}]^{\beta}\}, \tag{1.9}$$

gdzie:

$\tau_{b,a}$ – intensywność śladu feromonowego na krawędzi pomiędzy wierzchołkami b i a,
$\eta_{b,a}$ – informacja heurystyczna pochodząca z analizowanego problemu (widoczność miasta c z miasta a – odwrotność odległości),
β – parametr określający wagę relatywnej ważności śladu feromonowego oraz informacji heurystycznej.

Jeśli wylosowana liczba $q > q_0$, gdy mrówka k znajduje się w wierzchołku b, to prawdopodobieństwo jej przejścia do wierzchołka c jest dane wzorem:

$$p_{b,c}(t) = \begin{cases} \dfrac{\tau_{b,c}(t) \cdot [\eta_{b,c}]^{\beta}}{\sum\limits_{a \notin tabu_k} \tau_{b,a}(t) \cdot [\eta_{b,a}]^{\beta}}, & \text{dla } c \notin tabu_k, \\ 0, & \text{w przeciwnym razie,} \end{cases} \tag{1.10}$$

gdzie:

$tabu_k$ – lista wierzchołków odwiedzonych przez mrówkę k.

Innymi słowy, z pewnym prawdopodobieństwem, określanym przez parametr q_0, mrówka może albo wybrać kolejny krok zgodnie z probabilistycznym wyborem (wzór (1.10)), na który ma wpływ ślad feromonowy oraz informacja heurystyczna, albo zoptymalizuje jakość tego wyboru (wzór (1.9)). Wartości parametru q_0 bliższe wartości 1 faworyzują eksploatację nad eksploracją. Jeśli $q_0 = 0$, reguła przejścia do kolejnego wierzchołka jest tożsama z regułą w podstawowej wersji AS (gdy $\alpha = 1$).

Pseudokod systemu mrowiskowego został przedstawiony jako alg. 1.3, gdzie:

b, c – wierzchołki (miasta),
(b, c) – krawędź pomiędzy wierzchołkami b i c,
b_{k1} – miasto startowe mrówki k,
A – zbiór mrówek,
N – liczba miast,
L_k – długość trasy zbudowanej przez mrówkę k,
R_{best} – długość najkrótszej znalezionej trasy,

System mrówkowy z rankingiem

W systemie mrówkowym z rankingiem (ang. *rank-based Ant System*, AS_{rank}) [BHS97] wdrożony został mechanizm rangowej aktualizacji feromonu. Spośród wszystkich mrówek wybierana jest zadana liczba mrówek o najlepszych rozwiązaniach i tylko one mogą odłożyć ślad feromonowy na mapie feromonowej. Wartość tego śladu zależy od jakości zbudowanego przez daną mrówkę rozwiązania. Ponadto mrówka z globalnie najlepszym rozwiązaniem dostaje dodatkową wartość feromonu.

System mrówkowy MAX-MIN

W systemie mrówkowym MAX-MIN ($\mathcal{MM}$AS) [SH00] położono silny nacisk na eksploatację poprzez uwzględnianie podczas aktualizacji śladu feromonowego tylko tej mrówki, która uzyskała najlepszy wyniki. Może to być mrówka, która w danej iteracji znalazła najlepsze rozwiązanie (wersja lokalna) lub taka, która w dotychczasowym wykonaniu algorytmu znalazła najlepsze rozwiązanie (wersja globalna). Jednocześnie użyto mechanizmu ograniczania wpływu śladu feromonowego, by uniknąć przedwczesnej zbieżności.

Ograniczono możliwe wartości śladu feromonowego do zadanego z góry przedziału $\langle \tau_{min}, \tau_{max} \rangle$. Wprowadzenie dolnej granicy gęstości śladu feromonowego gwarantuje co najmniej minimalny poziom eksploracji. Zbyt duża wartość górnej granicy kumulacji feromonu lub jej brak może prowadzić do nieograniczonej kumulacji feromonu i szybkie przejście w fazę eksploatacji, a tym samym utknięcie w optimum lokalnym. Zbyt niska wartość tej granicy nie pozwala promować bardziej obiecujących rozwiązań i sprawia, że kolejne rozwiązania są w dużej mierze losowe. Domyślnie ślad feromonowy jest inicjowany w przybliżeniu wartością bliską maksymalnej dostępnej wartości.

Algorytm 1.3: SYSTEM MROWISKOWY

Wejście: macierz odległości pomiędzy miastami, $\tau_0, q_0, \beta, |A|$
Wyjście: najkrótsza znaleziona trasa

foreach (b, c) **do**
  $\tau_{b,c} \leftarrow \tau_0$
while *nie osiągnięto warunku zatrzymania* **do**
  foreach $k \in A$ **do**
    umieść k w mieście b_{k1}
    $tabu_k \leftarrow \varnothing + b_{k1}$
    $b_k \leftarrow b_{k1}$
  for $l \leftarrow 2$ ***to*** N **do**
    foreach $k \in A$ **do**
      wylosuj liczbę $q \in \langle 0, 1 \rangle$
      if $q \leqslant q_0$ **then**
        $c_k \leftarrow \arg \max_{a \notin tabu_k} \{[\tau_{b,a}] \cdot [\eta_{b,a}]^\beta\}$
      else
        $c_k \leftarrow$ wylosuj wierzchołek spośród $c \notin tabu_k$ zgodnie z prawdopodobieństwem: $p_{b,c} = \dfrac{\tau_{b,c} \cdot [\eta_{b,c}]^\beta}{\sum_{a \notin tabu_k} \tau_{b,a} \cdot [\eta_{b,a}]^\beta}$
      $tabu_k \leftarrow tabu_k + c_k$
    foreach $k \in A$ **do**
      $\tau^k_{b,c} \leftarrow (1 - \rho) \cdot \tau^k_{b,c} + \rho \cdot \tau_0$
      $b_k \leftarrow c_k$
  foreach $k \in A$ **do**
    $tabu_k \leftarrow tabu_k + b_{k1}$
    oblicz L_k
  $L_{best} = \min_{k \in A}(L_k)$
  foreach (b, c) **do**
    if $(b, c) \in L_{best}$ **then**
      $\tau^k_{b,c} \leftarrow (1 - \sigma) \cdot \tau^k_{b,c} + \dfrac{\sigma}{L_{best}}$
    else
      $\tau^k_{b,c} \leftarrow (1 - \sigma) \cdot \tau^k_{b,c}$
  if $L_{best} > R_{best}$ **then**
    $R_{best} = L_{best}$
return R_{best}

System mrówkowy „najlepszy-najgorszy”

W przypadku systemu mrówkowego „najlepszy-najgorszy” (ang. *Best-Worst*, BWAS) [CdVHM00] z jednej strony tylko najlepsza mrówka może na koniec cyklu zwiększyć wartość feromonu na mapie feromonowej, z drugiej zaś strony rozwiązanie najgorszej mrówki zostaje „ukarane” przez dodatkowe wyparowanie feromonu dla krawędzi zbudowanego przez nią rozwiązania. Dodatkowym mechanizmem wprowadzającym różnorodność badanych obszarów przestrzeni rozwiązań jest mutacja śladu feromonowego, która z pewnym prawdopodobieństwem wprowadza pewną zmianę wartości feromonu dla każdego wierzchołka. Możliwa jest również ponowna inicjalizacja mapy feromonowej w przypadku braku poprawy budowanych rozwiązań.

1.3.3 Zastosowania algorytmów optymalizacji mrowiskowej

Pierwszy raz system mrówkowy został zastosowany do problemu komiwojażera [DMC91, Dor92]. Sprawdza się on tam, gdzie potencjalne rozwiązania są grafami lub mogą jako takie zostać przedstawione. Gdy dla pewnych problemów brak dokładnych algorytmów lub algorytmy te są zbyt kosztowne obliczeniowo, algorytmy optymalizacji mrowiskowej mogą pomóc w osiągnięciu dobrych rezultatów w stosunkowo krótkim czasie. Z tego powodu aktualne zastosowania algorytmu optymalizacji mrowiskowej obejmują wiele dziedzin i zagadnień. Są to między innymi:

- problem komiwojażera [DMC91, Dor92],
- kolorowanie grafów [CH97],
- problem marszrutyzacji [GTA99],
- kwadratowy problem przydziału [MC99],
- trasowanie w sieciach komunikacyjnych [DCD99],
- planowanie [MMS02],
- eksploracja danych [KB16, PLF02],
- muzyka [GMS04],
- bioinformatyka [SH05],
- przetwarzanie obrazu [JQDCJA09],
- wyszukiwanie informacji w Internecie [BP09],

- sieci społecznościowe [MHM17],
- optymalizacja GPU [PM18].

Jednak nie stosowano, jak dotąd, tego podejścia do kryptoanalizy szyfrów strumieniowych.

Rozdział 2

Kryptologia

Sztuka szyfrowania jest tak stara jak umiejętność zapisania i przesłania wiadomości. Już starożytni Spartanie posługiwali się szyfrem oraz prymitywnym, ale skutecznym w tamtych czasach, urządzeniem szyfrującym [Coh95, Kah04]. We współczesnym ujęciu, zwłaszcza w kontekście przetwarzania wiadomości w systemach informatycznych, mówi się o kryptologii. Jest to dziedzina wiedzy zajmująca się komunikacją zabezpieczoną przed dostępem osób innych niż nadawca i odbiorca. Jedną z gałęzi kryptologii jest kryptografia zajmująca się szyfrowaniem informacji. Z założenia jest to takie przekształcenie wiadomości, by niemożliwe (lub bardzo trudne) było odczytanie jej przez każdego, kto nie posiada odpowiedniego klucza. Po drugiej stronie znajduje się kryptoanaliza, będąca zarówno przeciwieństwem, jak i uzupełnieniem kryptografii. Celem kryptoanalizy jest złamanie szyfru, co może być rozumiane w szerokim zakresie: od ujawnienia wartości klucza pozwalające na odszyfrowanie wszystkich wiadomości zaszyfrowanych tym kluczem po jedynie rozróżnienie szyfrogramu od ciągu wygenerowanego losowo. Kryptoanalitycy to nie tylko niepowołane osoby pragnące odczytać nie przeznaczone dla nich wiadomości. Kryptoanalitycy badają powstające i istniejące algorytmy szyfrujące, chcąc znaleźć ich słabe punkty, aby kolejne proponowane rozwiązania były coraz bezpieczniejsze.

2.1 Podstawowe pojęcia

Nazwa „kryptologia" pochodzi z języka greckiego. Jest to złączenie słów: $\kappa\rho\upsilon\pi\tau\acute{o}\zeta$ – *kryptos* – ukryty oraz $\lambda\acute{o}\gamma o\zeta$ – *logos* – słowo. Podobnie rzecz ma się z „kryptografią" oraz „kryptoanalizą". Drugi człon tych słów oznacza odpowiednio: $\gamma\rho\acute{\alpha}\varphi\omega$ – *gráfo* – pisać i $\acute{\alpha}\nu\alpha\lambda\acute{\upsilon}\varepsilon\iota\nu$ – *analýein* – rozluźnić, rozsupłać [Sim17].

Z kryptologią związane są następujące terminy:

- tekst jawny, tekst otwarty (ang. *plaintext*) – wiadomość przed zaszyfrowaniem; oznaczany jako P,
- przestrzeń tekstów jawnych – zbiór wszystkich dopuszczalnych tekstów jawnych;
- szyfrogram, tekst zaszyfrowany, kryptogram (ang. *ciphertext*) – wiadomość zaszyfrowana; oznaczany jako C,
- przestrzeń szyfrogramów – zbiór wszystkich możliwych szyfrogramów;
- szyfrowanie, przekształcenie szyfrujące (ang. *encryption*) – przekształcenie wiadomości (tekstu jawnego), które utajnia jej treść; oznaczane jako E,
- deszyfrowanie, przekształcenie deszyfrujące, przekształcenie rozszyfrowujące (ang. *decryption*) – przekształcenie szyfrogramu, które odtwarza oryginalną wiadomość (tekst jawny); proces odwrotny do szyfrowania; oznaczane jako D,
- schemat szyfrowania, system kryptograficzny, algorytm szyfrujący, szyfr (ang. *cipher*) – funkcja matematyczna (lub para funkcji) pozwalająca na szyfrowanie i deszyfrowanie,
- klucz (ang. *key*) – pomocnicza informacja umożliwiająca szyfrowanie i deszyfrowanie; oznaczany jako K,
- przestrzeń kluczy – zbiór wszystkich możliwych wartości, jakie może przyjąć klucz w danym algorytmie szyfrującym,
- strumień szyfrujący, strumień kluczujący (ang. *keystream*) – strumień (najczęściej bitów) wygenerowany z klucza, służący do szyfrowania tekstu jawnego w szyfrach strumieniowych; oznaczany jako κ,
- klucz jawny i tajny, klucz publiczny i prywatny – para skorelowanych kluczy używana w algorytmach asymetrycznych; oznaczane jako K_{PU} (klucz publiczny) i K_{PR} (klucz prywatny).

Tekst jawny i szyfrogram są zapisane przy użyciu pewnego alfabetu. Alfabet to skończony zbiór symboli. Zatem tekst jawny oraz szyfrogram będą ciągami symboli określonego alfabetu. Najczęściej, choć nie zawsze, alfabety do opisu tekstu jawnego i szyfrogramu są tożsame. Rozpatrywany tekst jawny może być zwykłym tekstem w danym języku i wtedy dla języka polskiego alfabet tekstu jawnego będzie zawierał 32 litery, spację i znaki interpunkcyjne. Rozpatrywany tekst jawny może być również wstępnie zakodowany

w jakiejś formie, na przykład dźwięk i grafika zakodowane w formie binarnej i wtedy alfabet tekstu jawnego będzie składał się tylko z dwóch znaków: 0 i 1. W szczególności każdy alfabet może być zakodowany w postaci alfabetu binarnego [Kob06, MVOV05].

Szyfrowanie (wzór (2.1)) i deszyfrowanie (wzór (2.2)) w systemach symetrycznych można zapisać w następujący sposób:

$$E(K, P) = C, \tag{2.1}$$

$$D(K, C) = P. \tag{2.2}$$

Przy czym, aby uzyskać prawidłowy algorytm szyfrujący, funkcje te dla wszystkich dopuszczalnych tekstów jawnych muszą charakteryzować się następującą właściwością:

$$D(K, E(K, P)) = P. \tag{2.3}$$

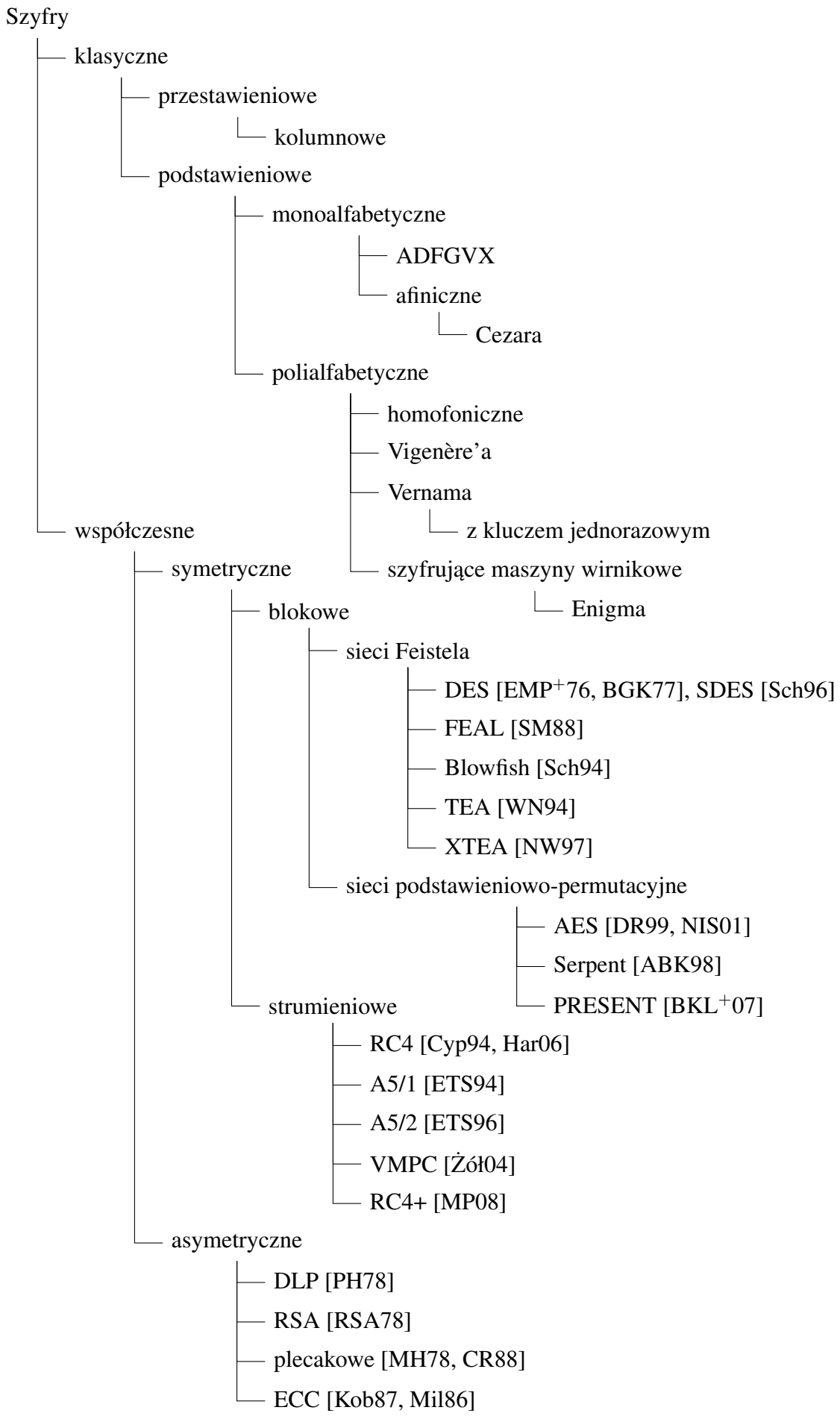
Dla systemów asymetrycznych powyższe równania przyjmą następującą postać:

- szyfrowanie:
$$E(K_{PU}, P) = C, \tag{2.4}$$
- deszyfrowanie:
$$D(K_{PR}, C) = P. \tag{2.5}$$

Ponownie, aby uzyskać prawidłowy algorytm szyfrujący, funkcje te dla wszystkich dopuszczalnych tekstów jawnych muszą charakteryzować się następującą właściwością:

$$D(K_{PR}, E(K_{PU}, P)) = P. \tag{2.6}$$

Schemat szyfrowania i deszyfrowania przedstawiono na rys. 2.2. Szyfry opracowane i używane przed erą komputerów nazywane są szyframi klasycznymi (historycznymi), natomiast wykorzystujące przetwarzanie komputerowe to szyfry współczesne. Zarówno w szyfrach klasycznych, jak i współczesnych można wyróżnić permutacje i podstawienia, przekształcenia liniowe i nieliniowe. Podział i zestawienie szyfrów omawianych w niniejszej rozprawie znajduje się na rys. 2.1.



Rysunek 2.1: Klasyfikacja szyfrów wymienionych w rozprawie

(a) w algorytmach symetrycznych

(b) w algorytmach asymetrycznych

Rysunek 2.2: Szyfrowanie i deszyfrowanie [Sch02]

2.2 Podział szyfrów

Wiele z powszechnie stosowanych obecnie algorytmów szyfrujących opiera swoje działanie na dwóch podstawowych operacjach, którymi są przestawienia i podstawienia. Ponieważ te elementarne operacje z szyfrów klasycznych stanowią również podstawę w budowie szyfrów współczesnych, to właśnie szyfry klasyczne często są pierwszym celem dla nowopowstających ataków. Pierwsza grupa szyfrów klasycznych to szyfry przestawieniowe, czyli takie, gdzie wszystkie litery tekstu jawnego pozostają niezmienione, a zmienia się jedynie kolejność ich występowania w tekście po zaszyfrowaniu.

Następną grupą szyfrów klasycznych są szyfry podstawieniowe. W szyfrogramach wygenerowanych przez te systemy kolejność znaków pozostaje bez zmian, natomiast są one zamieniane na inne znaki zgodnie z przyjętą metodą szyfrowania, określonym alfabetem i wybranym kluczem.

Zakładając, że w takim systemie jednemu znakowi tekstu jawnego odpowiada jeden znak szyfrogramu i jest on szyfrowany za pomocą jednego znaku klucza, to – gdy klucz jest długości tekstu poddawanego szyfrowaniu i jest wygenerowany w całości losowo – taki rodzaj szyfrowania nazywany jest szyfrem/systemem z kluczem jednorazowym (ang. *one-time pad*, OTP) [Kah04, Sin01]. Jeśli dodatkowo dla każdej wiadomości używany jest inny losowo wygenerowany klucz, udowodniono, że takiego szyfru nie można złamać. Jego wadą jest właśnie wymóg generowania ogromnej długości kluczy (lub zbiorów kluczy) oraz problem bezpiecznego przekazania ich drugiej stronie. Z tego względu rzadko jest stosowany w praktyce, tylko dla wyjątkowo tajnych wiadomości, w kanałach o małej przepustowości [MVOV05, Sch02]. Taki szyfr jest również czasem nazywany szyfrem

Vernama, choć tak naprawdę szyfr Vernama pozwala na powtórzenie lub zapętlenie klucza, a to z kolei obniża bezpieczeństwo i sprawia, że kryptoanaliza tak przesłanej wiadomości jest możliwa.

Ostatnią grupę szyfrów klasycznych stanowią szyfrujące maszyny wirnikowe. Przykładem takiej maszyny jest znana na całym świecie Enigma, używana w trakcie II wojny światowej.

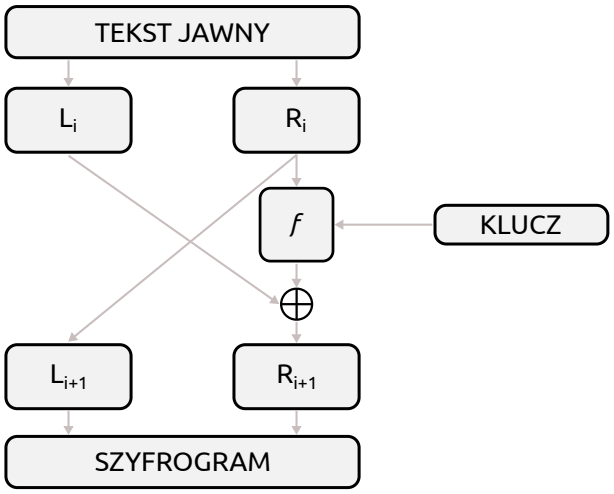
Współczesne algorytmy szyfrujące dzielą się na dwie główne grupy: algorytmy symetryczne i asymetryczne. Pierwsze z nich do szyfrowania i deszyfrowania korzystają z tego samego klucza (lub dwóch kluczy, ale można łatwo obliczyć drugi z nich na podstawie pierwszego). Dodatkowo szyfry symetryczne dzielą się na szyfry blokowe oraz strumieniowe.

Działanie szyfrów blokowych polega na szyfrowaniu danych podzielonych na pewne bloki o ustalonej długości. Każdy blok z osobna zostaje poddany pewnemu przekształceniu i w wyniku tego powstaje blok szyfrogramu. Dodatkowo często stosuje się wiązanie bloków, co oznacza, że tekst jawny lub szyfrogram z poprzedniego bloku ma wpływ na przetwarzanie bloku następnego. Pozwala to zapobiec ujawnieniu informacji, gdy dwa identyczne bloki tekstu jawnego zostaną przekształcone w ten sam szyfrogram oraz znacząco utrudnia ewentualną kryptoanalizę. Szyfry blokowe mają zastosowanie w:

- szyfrowaniu dysków twardych i innych nośników danych (pendrive, płyty CD i DVD, itp.)
- szyfrowaniu plików,
- szyfrowaniu danych w bazach danych,
- szyfrowaniu danych przesyłanych i przechowywanych pocztą elektroniczną oraz przez komunikatory,
- przewodowej transmisji danych (SSL/TLS, SSH, w tym usługi bankowości elektronicznej).

Jedną ze struktur stosowanych w budowie blokowych algorytmów szyfrujących jest sieć Feistela (rys. 2.3). Pozwala ona na szyfrowanie i deszyfrowanie tekstów za pomocą tego samego algorytmu, pomimo tego iż właściwa funkcja szyfrująca f nie musi być odwracalna. Taka konstrukcja znacznie uprościła tworzenie algorytmów kryptograficznych, ponieważ odwracalność funkcji f nie musi być zapewniona.

Inną, bardziej ogólną możliwością skonstruowania szyfru blokowego jest sieć podstawieniowo-permutacyjna (sieć SP, SPN, ang. *substitution-permutation network*), pokazana na rys. 2.4. Składa się ona z na przemian występujących bloków przestawień



Rysunek 2.3: Schemat jednej rundy sieci Feistela (L, R – dwa bloki tekstu równej długości, i – numer rundy, f – właściwy algorytm szyfrujący)

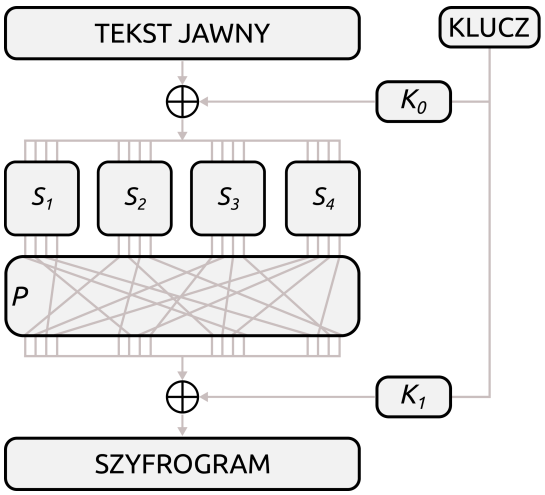
(permutacji, P-bloków) i podstawień (skrzynek podstawieniowych, S-skrzynek, S-bloków). Połączenie obu rodzajów bloków jest konieczne ze względów zarówno praktycznych, jak i ze względów bezpieczeństwa. Przekształcenia liniowe są stosowane, ponieważ cechuje je łatwa implementacja sprzętowa i programowa, a także łatwość analizy ich własności. Z drugiej jednak strony charakteryzują się dużą liniowością, co ułatwia kryptoanalizę. Dlatego łączone są z przekształceniami nieliniowymi. Te z kolei są trudniejsze do implementacji oraz analizy własności, więc są to raczej niewielkie elementy. Są jednak używane, ponieważ znacząco utrudniają kryptoanalizę. Pojedynczy blok permutacji lub podstawień nie ma zbyt wiele kryptograficznej siły, ale dobrze skonstruowany szyfr z wieloma naprzemiennymi rundami permutacji i podstawień stanowi dobre zabezpieczenie. Sieć SPN w realizacji potrafi być podobna do sieci Feistela, jednak użyte w niej S-bloki muszą być odwracalne, podczas gdy funkcja użyta w sieci Feistela odwracalna być nie musi. Są też inne różnice działania, które sprawiają, że każdy rodzaj sieci nadaje się trochę bardziej do innych zastosowań [Fei73, Fei74].

Drugą grupą szyfrów symetrycznych są szyfry strumieniowe. W przeciwieństwie do sposobu działania szyfrów blokowych, działanie szyfrów strumieniowych polega na szyfrowaniu nie bloków danych, a pojedynczych bitów, rzadziej bajtów lub znaków. Szyfry takie mają zastosowanie w:

- transmisji danych przez karty płatnicze (w tym zbliżeniowe oraz płatności telefonem),
- transmisji danych przez karty elektroniczne (w tym zbliżeniowe), na przykład karty dostępu,
- transmisji głosowej w telefonii komórkowej,
- transmisji danych przesyłanych przez cyfrowe radio i telewizję,

- bezprzewodowej transmisji danych (przykładowo drogą radiową, Bluetooth, WiFi),
- przewodowej transmisji danych (SSL/TLS, SSH, w tym usługi bankowości elektronicznej).

Dokładny opis szyfrów strumieniowych znajduje się w podrozdz. 2.4.



Rysunek 2.4: Schemat jednej rundy sieci podstawieniowo-permutacyjnej (S_i – blok podstawieniowy, P – blok permutacyjny, K_i – podklucz rundy)

Szyfry asymetryczne, w przeciwieństwie do szyfrów symetrycznych, wymagają istnienia pary skorelowanych kluczy, z czego jeden z nich jest jawny (publiczny), a drugi tajny (prywatny). Wiadomość szyfruje się pierwszym kluczem, natomiast by odszyfrować ją należy użyć drugiego klucza. Z założenia wydedukowanie jednego klucza na podstawie drugiego jest nieopłacalne obliczeniowo. Oznacza to, że w teorii jest to możliwe, w praktyce jednak koszt obliczeń jest większy od wartości ukrytej informacji lub czas potrzebny na wykonanie obliczeń jest większy niż czas użyteczności ukrytej informacji.

Szczególnym przypadkiem są algorytmy szyfrujące oparte na problemie plecakowym, który jest jednym z problemów NP-zupełnych [Kob06]. Ponieważ powstały one dopiero w dobie komputerów, nie można ich zaliczyć do szyfrów klasycznych; z drugiej zaś strony nie zostały wykorzystane do stworzenia współczesnego szyfru. Upraszczając można powiedzieć, że taki system kryptograficzny (na przykład Merkle-Hellman [MH78], Chor-Rivest [CR88]) opiera się na jednym problemie plecakowym, który może być przedstawiony w dwóch wersjach – jedna jest łatwa i służy jako klucz prywatny; druga jest trudna, wyliczona na podstawie pierwszej i pełni rolę klucza publicznego. Bezpieczeństwo szyfrów opartych na tym problemie bazuje na trudności rozwiązania trudnej wersji rozpatrywanego problemu plecakowego. Złożoność obliczeniowa na poziomie NP-zupełnym oraz szybkość operacji szyfrowania i deszyfrowania zdecydowanie lepsza w porównaniu z innymi szyframi asymetrycznymi sprawiły, że tego typu szyfry wydawały się bardzo

atrakcyjne. Jednakże okazały się one wrażliwe na pewne typy ataków, co ostatecznie sprawiło, iż nie są to algorytmy szyfrujące używane w praktyce [Odl90].

Używanym w rzeczywistych zastosowaniach szyfrem asymetrycznym jest algorytm RSA (od nazwisk twórców: Ron Rivest, Adi Shamir, Leonard Adleman). Jego bezpieczeństwo opiera się na trudności rozkładu na czynniki pierwsze dużych liczb złożonych. W zależności od oczekiwanego poziomu bezpieczeństwa obecnie zaleca się klucz o długości 1024–4096 bitów. W praktycznych zastosowaniach można również spotkać się z systemami opartymi na problemie logarytmu dyskretnego (ang. *Discrete Logarithm Problem*, DLP) oraz na krzywych eliptycznych (ang. *Elliptic Curve Cryptography*, ECC). ECC oferuje bezpieczeństwo porównywalne do RSA przy znacznie krótszych kluczach.

2.3 Kryptoanaliza

Celem kryptoanalizy jest znalezienie słabości w istniejących oraz projektowanych algorytmach szyfrujących. Z jednej strony jest to poszukiwanie luk w zabezpieczeniach przez osoby chcące przejąć tajną wiadomość nieprzeznaczoną dla nich, z drugiej jest to badanie przez naukowców rozwiązań kryptograficznych w celu poprawy ich bezpieczeństwa i jakości.

Podstawowa zasada kryptologii mówi, że algorytm szyfrujący powinien być bezpieczny, nawet jeśli wszystkie szczegóły jego budowy oraz działania są znane. Jedyną nieznaną dla osoby postronnej informacją powinien być klucz i nieznajomość wyłącznie jego powinna być wystarczająca, by utajnić odbywającą się komunikację. Założenie to sformułował pod koniec XIX wieku Auguste Kerckhoffs [Ker83]. Zatem, zgodnie z zasadą Kerckhoffsa, należy założyć, że kryptoanalityk zna wszystkie szczegóły algorytmu szyfrowania i jego implementacji. Takie właśnie założenie zostało przyjęte w ataku kryptoanalitycznym przedstawionym w ramach niniejszej rozprawy.

Poszukiwać słabości w algorytmach szyfrujących można na wiele różnych sposobów. Najmocniejszym z nich jest atak z szyfrogramem – agresor mając jedynie szyfrogram jest w stanie odzyskać z niego klucz i/lub tekst jawny. Po drugiej stronie skali znajduje się atak rozróżniający. W tym przypadku mając dany strumień szyfrujący, można uzyskać jedynie informację na temat tego, czy strumień ów został wygenerowany przez konkretny algorytm szyfrujący czy też jest on wygenerowany w sposób losowy. Jednak każdy rodzaj ataku jest równie ważny. W kryptoanalizie ujawnienie wiedzy na temat wartości choćby jednego bitu jest uznawane za naruszenie bezpieczeństwa. Ponadto z czasem szyfry nigdy nie stają się bezpieczniejsze i historia kryptologii pełna jest przypadków, gdy zaczynając od najprostszych rodzajów ataków, z biegiem lat i kolejnych odkryć, złamano kolejne szyfry.

Ataki na szyfry opisane w przeglądzie literatury (rozdz. 3) można pogrupować następująco [MVOV05, Sch02, Sta12]:

1. Ze względu na to, do jakiego rodzaju informacji kryptoanalityk ma dostęp wyróżnia się:

   - atak z szyfrogramem (ang. *ciphertext-only attack*, COA lub *known ciphertext attack*): kryptoanalityk dysponuje szyfrogramem pewnej wiadomości,
   - atak z wybranym szyfrogramem (ang. *chosen ciphertext attack*, CCA): kryptoanalityk może zdeszyfrować dowolny spreparowany przez siebie szyfrogram,
   - atak ze znanym tekstem jawnym (ang. *known plaintext attack*, KPA): kryptoanalityk dysponuje zarówno szyfrogramem, jak i odpowiadającym mu tekstem jawnym; przykładami tego typu ataku są:
     - kryptoanaliza liniowa (ang. *linear cryptanalysis*),
     - atak korelacyjny (ang. *correlation attack*),
   - atak z wybranym tekstem jawnym (ang. *chosen plaintext attack*, CPA): kryptoanalityk ma możliwość wybrania tekstu jawnego i ma również dostęp do odpowiadającego mu szyfrogramu; przykłem tego typu ataku jest:
     - kryptoanaliza różnicowa (ang. *differential cryptanalysis*, DC).

   Ataki z tej grupy będą to najczęściej bezpośrednie ataki na konstrukcję algorytmu szyfrującego.

2. Wyróżniana jest również klasa ataków kanałem bocznym (ang. *side-channel attack*, SCA), w tym:

   - analiza czasu wykonywania,
   - analiza poboru mocy,
   - analiza dźwięku,
   - analiza błędów (np. różnicowa analiza błędów, ang. *differential fault analysis*).

   Będą to najczęściej ataki na urządzenia szyfrujące lub konkretne implementacje.

3. Ze względu na cel działania wyróżnia się ataki, których celem jest:

   - kompromitacja klucza,
   - ujawnienie stanu wewnętrznego algorytmu szyfrującego (ang. *state recovery attack*),

- ujawnienie tekstu jawnego (bez znajomości klucza),
- odróżnienie od wartości losowych (atak rozróżniający, ang. *distinguishing attack*).

Warto podkreślić, że sukcesem będzie również nawet częściowe osiągnięcie jednego z powyższych celów, przykładowo ujawnienie nie całego tekstu jawnego, a jedynie jego fragmentu.

4. W ramach powyższych ataków można również przeprowadzić atak siłowy (ang. *brute force attack*) lub inaczej przeszukiwanie wyczerpujące (ang. *exhaustive key search*).

Należy zaznaczyć, że powyższa lista, jak i zestawienie z rys. 2.1, nie przedstawiają problemu w sposób wyczerpujący, gdyż ta tematyka nie wchodzi w zakres rozprawy.

2.4 Szyfry strumieniowe

Jak zostało powiedziane w podrozdz. 2.2 szyfry strumieniowe należą do grupy szyfrów symetrycznych. Tekst jawny zakodowany alfabetem binarnym jest zazwyczaj dzielony na bity (ewentualnie bajty lub słowa), a następnie szyfrowany kolejnym bitem (ewentualnie bajtem lub słowem) strumienia szyfrującego. Najczęściej strumień ten bit po bicie jest dodawany do tekstu jawnego, wykorzystując alternatywę wykluczającą, czyli operację XOR, oznaczaną symbolem $\oplus$. W ten sposób uzyskana zostaje wiadomość w zaszyfrowanej formie:

$$P_m \oplus \kappa_m = C_m, \tag{2.7}$$

gdzie:

P_m – m-ty bit tekstu jawnego,
κ_m – m-ty bit strumienia szyfrującego,
C_m – m-ty bit szyfrogramu.

Strumień szyfrujący powinien być co najmniej takiej długości jak tekst jawny. Aby uzyskać z powrotem tekst jawny, trzeba wykonać analogiczną operację w odwrotnym kierunku: dodać przy użyciu funkcji XOR strumień szyfrujący oraz szyfrogram:

$$C_m \oplus \kappa_m = P_m. \tag{2.8}$$

Przykładowo:

$$\begin{array}{lcl} P: & & 10011101011100011011011000110111... \\ \kappa: & \oplus & 11100111110101000011110100101100... \\ \hline C: & & 01111010101001011000101100011011... \end{array}$$

Oczywiście szyfrowanie i deszyfrowanie z użyciem szyfrów strumieniowych spełnia właściwość z wzoru (2.3) (str. 34), ponieważ:

$$C_m \oplus \kappa_m = (P_m \oplus \kappa_m) \oplus \kappa_m = P_m \oplus \kappa_m \oplus \kappa_m = P_m \oplus 0 = P_m. \tag{2.9}$$

W przypadku gdy strumień szyfrujący będzie generowany przez pewien generator liczb pseudolosowych, wynikiem będzie szyfr strumieniowy. Gdy strumień szyfrujący będzie generowany przez generator prawdziwie losowy i użyty do zaszyfrowania tylko jednej wiadomości, wynikiem będzie szyfr OTP – jak już wspomniano jest to jedyny szyfr, dla którego zostało udowodnione, iż niemożliwe jest jego złamanie, nawet poprzez atak siłowy, czyli przeszukanie wyczerpujące przestrzeni kluczy.

W szyfrowaniu strumieniowym przeważnie nie używa się bezpośrednio klucza K, lecz służy on do wygenerowania dłuższego strumienia szyfrującego κ. Jest tak ze względu na to, że okres powtarzalności takiego strumienia jest znacznie dłuższy (np. $\sim 2^{500}$) niż okres powtarzalności samego klucza (długości obecnie zalecanych kluczy są z przedziału $2^7 - 2^{12}$ bitów). W związku z tym istotne jest właściwe skonstruowanie algorytmu generującego strumień szyfrujący κ.

Działanie większości szyfrów strumieniowych jest podzielone na dwa etapy:

1. KSA (ang. *Key Scheduling Algorithm*) – faza, w której w wewnętrznej strukturze algorytmu rozpraszana jest wartość tajnego klucza K oraz ewentualnie jawnego wektora inicjacyjnego (ang. *initialization vector*, IV).

2. PRGA (ang. *Pseudo-Random Generation Algorithm*) – faza generowania strumienia szyfrującego κ; w tej fazie ani klucz, ani wektor IV nie są już używane.

Określenie stanu wewnętrznego algorytmu szyfrującego bezpośrednio po KSA pozwoli na wygenerowanie późniejszego strumienia szyfrującego bez potrzeby znajomości tajnego klucza, a co za tym idzie, bez potrzeby znajomości klucza, będzie można odszyfrować tajną wiadomość. W tym miejscu zatem można przeprowadzić atak mający na celu ujawnienie stanu wewnętrznego algorytmu szyfrującego (ang. *state recovery attack*). Ze względu na nieistotność fazy KSA w niniejszych rozważaniach, algorytmy dystrybucji klucza zostaną pominięte w opisach algorytmów szyfrujących.

Celem niniejszej rozprawy jest stworzenie podejścia przyspieszającego proces kryptoanalizy oraz niezależnego od specyfiki algorytmu szyfrującego. Stąd wykorzystanie algorytmów miękkich może być bardzo korzystnym wyjściem.

Opisane w kolejnych punktach szyfry strumieniowe mają podobną strukturę wewnętrzną. Ich główną część stanowi permutacja 256 liczb całkowitych, od 0 do 255. Wszystkie produkują w każdym takcie jeden bajt strumienia szyfrującego (w fazie PRGA). Następnie bajt strumienia szyfrującego jest dodawany za pomocą funkcji XOR do kolejnego bajtu tekstu jawnego, czego efektem jest bajt szyfrogramu. Z każdym wyprodukowanym taktem stan wewnętrzny tych algorytmów szyfrujących się zmienia. Oprócz tych podobieństw fazy rozpraszania klucza i generowania strumienia są różne dla każdego szyfru.

W kolejnych punktach oraz w dalszej części rozprawy przyjęto następujące oznaczenia:

S – 256-bajtowa tablica zawierająca permutację liczb całkowitych $\{0, 1, ..., 255\}$,
i, j – indeksy 8-bitowe całkowitoliczbowe,
L – długość wiadomości w bajtach,
$+$ – dodawanie $\mod 256$,
$\oplus$ – XOR,
$i \ll x$ ($i \gg x$) – przesunięcie bitowe o x w lewo (prawo).

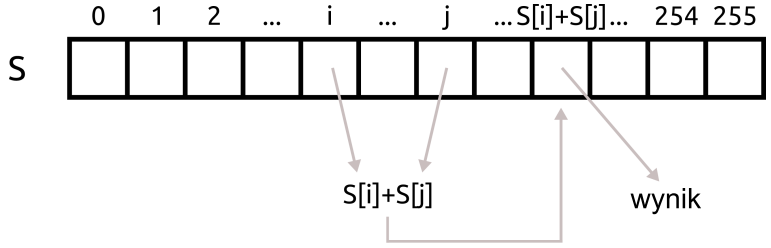
2.4.1 Szyfr RC4

RC4 to szyfr strumieniowy zaprojektowany przez Rona Rivesta, w RSA Security w 1987 roku. Nazwa szyfru pochodzi od nazwiska twórcy: RC4 to akronim angielskiego wyrażenia *Rivest-Cipher 4*. Na początku był objęty tajemnicą handlową, po kilku latach algorytm został anonimowo wysłany na listę mailingową Cypherpunks [Cyp94]. Szyfr RC4 posiada zmienną długość klucza (do 2048 bitów), który wyznacza permutację początkową liczb w 256-elementowej tablicy S. Szyfr RC4 może znajdować się w jednym z $256! \cdot 256 \cdot 256 \approx 2^{1700}$ możliwych stanów (256-elementowa permutacja oraz dwie 8-bitowe zmienne). Strumień szyfrujący jest długości tekstu jawnego i jest od niego niezależny. Algorytm został zaimplementowany między innymi w SSL/TLS (ang. *Secure Socket Layer/Transport Layer Security*) do szyfrowania ruchu w Internecie oraz w WEP (ang. *Wired Equivalent Privacy*) i WPA (ang. *WiFi Protected Access*) do zabezpieczenia sieci bezprzewodowych [Har06].

W swoim czasie szyfr RC4 był prawdopodobnie najbardziej powszechnie stosowanym szyfrem strumieniowym na świecie. Jego popularność wynika z łatwej możliwości implementacji zarówno programowej, jak i sprzętowej, a także z niezwykłej szybkości działania. Pseudokod szyfru strumieniowego RC4 przedstawia alg. 2.1, a schemat szyfrowania za jego pomocą – rys. 2.5. Procedura generowania strumienia szyfrującego składa się z czterech instrukcji wykonywanych w pętli. Te instrukcje to zmiana wartości

Algorytm 2.1: FAZA PRGA SZYFRU RC4

```
1 i ← 0
2 j ← 0
3 for b ← 1 to L do
4 |   i ← i + 1
5 |   j ← j + S[i]
6 |   zamień (S[i], S[j])
7 |   wynik ← S[ S[i] + S[j] ]
```



Rysunek 2.5: Schemat szyfrowania RC4

dwóch indeksów (z czego jeden jest inkrementowany), zamiana miejscami dwóch elementów permutacji oraz odczytanie wynikowego bajtu dla danej iteracji pętli.

Znane ataki

Przeszukiwanie wyczerpujące przestrzeni kluczy wymaga sprawdzenia 2^{2048} możliwości. Sprawdzenie wszystkich możliwych początkowych stanów wewnętrznych pozwoli zmniejszyć liczbę możliwości do 2^{1684} $(256!)$. Szyfr RC4 przez 30 lat istnienia został poddany wielorakiej kryptoanalizie [ABP+13, GMPS12, IOWM14, Mir02, PPS15]. Rezultaty otrzymane za pomocą algorytmów genetycznych, optymalizacji stadnej cząsteczek oraz symulowanego wyżarzania są opisane w podrozdz. 3.3. Najlepszym z algorytmów metaheurystycznych okazał się algorytm genetyczny. Benjamin Ferriman i Charlie Obimbo szacują, że poznanie stanu początkowego fazy PRGA szyfru RC4 wymaga sprawdzenia 2^{124} możliwości.

Najważniejszym z ataków jest przedstawiony przez Mathy'ego Vanhoefa i Franka Piessensa [VP15]. Pokazali oni atak na WPA-TKIP (ang. *Wi-Fi Protected Access – Temporal Key Integrity Protocol*), który może być wykonany w ciągu godziny. Ci sami autorzy zaprojektowali także praktyczny atak doprowadzający w ciągu 75 godzin do odzyskania ciasteczka (ang. *cookie*) protokołu TLS, używanego w HTTPS ze wskaźnikiem sukcesu 94%, z użyciem 2^{30} szyfrogramów. Są to ataki wykorzystujące słabości protokołów, ale również krótko- i długoterminowe odchylenia generowanego przez szyfr RC4 strumienia szyfrującego w stosunku do strumienia o rozkładzie jednostajnym. Warianty tej metody mogą zostać wykorzystane również w innych miejscach, gdzie używany jest szyfr RC4.

2.4.2 Szyfr VMPC

VMPC to funkcja oraz szyfr strumieniowy opublikowany przez Bartosza Żółtaka w 2004 roku [Żół04]. Nazwa jest skrótem angielskiego określenia *Variably Modified Permutation Composition*, czyli „zmiennie modyfikowane złożenie permutacji". Z oprogramowania szyfrującego z użyciem szyfr VMPC, to jest z programu VMPCrypt napisanego przez tego samego autora, korzystają na użytek wewnętrzny polskie przedsiębiorstwa, instytucje oraz osoby prywatne.

Definicja funkcji VMPC dla n-elementowej permutacji f jest następująca:

$$g(x) = VMPC(f(x)) = f(f(f(x)) + 1) \mod n. \tag{2.10}$$

Zgodnie z deklaracjami autora, szyfr VMPC oferuje znacznie lepsze właściwości statystyczne niż algorytm RC4. Stan wewnętrzny szyfru VMPC może przyjmować $256! \cdot 256^2 \approx 2^{1700}$ wartości (256-elementowa permutacja oraz dwie 8-bitowe zmienne). Początkowy stan wewnętrzny jest obliczany na podstawie tajnego klucza oraz jawnego wektora inicjacyjnego. Długość klucza, podobnie jak w RC4, jest zmienna (tu od 128 do 512 bitów). Każdorazowo po wygenerowaniu bajtu strumienia szyfrującego stan wewnętrzny ulega zmianie. Bezpieczeństwo szyfru opiera się na założeniu, że funkcja $g(x)$ jest jednokierunkowa.

Pseudokod szyfru strumieniowego VMPC przedstawia alg. 2.2. Podobnie jak w przypadku poprzedniego szyfru procedura generowania strumienia szyfrującego przy użyciu szyfru VMPC składa się z czterech instrukcji wykonywanych w pętli tak długo, jak potrzebny jest kolejny bajt do szyfrowania tekstu jawnego. Ponownie dla każdej iteracji pętli są to: zmiana wartości dwóch indeksów (w tym inkrementacja jednego), zamiana miejscami dwóch elementów permutacji i odczytanie wynikowego bajtu. Przy czym zmiana wartości indeksu j oraz obliczanie kolejnego bajtu strumienia szyfrującego są realizowane w odmienny sposób. Również kolejność tych instrukcji jest częściowo zmieniona.

Algorytm 2.2: FAZA PRGA SZYFRU VMPC

```
1 i ← 0
2 for b ← 1 to L do
3 |   j ← S[ j + S[i] ]
4 |   wynik ← S[S[S[j]]+1]
5 |   zamień (S[i], S[j])
6 |   i ← i + 1
```

Znane ataki

Atak brutalny wymaga sprawdzenia 2^{512} możliwych kluczy. Znana autorowi metoda odwrócenia funkcji f, a co za tym idzie złamania szyfru VMPC, wymaga średnio 2^{260} ($\approx 10^{78}$) operacji [Żół04]. Jednokierunkowość funkcji VMPC badał Kamil Kulesza [Kul06]. Opublikowane do tej pory ataki [LHZW12, Max05, Sar15, TSK+05] są atakami rozróżniającymi losowy strumień bitów od strumienia generowanego przez algorytm szyfrujący. Same w sobie nie pozwalają na bezpośrednie określenie klucza czy odszyfrowanie wiadomości. Najlepszy z tych ataków [Sar15] wymaga 2^{24} bajtów strumienia szyfrującego.

2.4.3 Szyfr RC4+

Algorytm szyfrujący RC4+ został zaproponowany w 2008 roku przez Subhamoy Maitra i Goutam Paul [MP08]. Jest on wersją algorytmu RC4 zmodyfikowaną w celu usunięcia słabości poprzednika. Stan wewnętrzny szyfru RC4+ stanowi permutacja 256 liczb oraz dwie zmienne 8-bitowe, zatem jak poprzednio w danej chwili algorytm może przyjąć jedną z 2^{1700} możliwości ($256! \cdot 256^2$; 256-elementowa permutacja oraz dwie 8-bitowe zmienne). Z kolei klucz i wektor inicjacyjny (IV) mogą mieć dowolną długość (klucz do 2048 bitów, IV do 1024 bitów).

Po zainicjowaniu stanu wewnętrznego kluczem oraz wektorem inicjacyjnym algorytm będzie działał zgodnie z krokami przedstawionymi w alg. 2.3. Instrukcje zmiany wartości dwóch indeksów oraz zamiany miejscami dwóch elementów permutacji są tożsame z tymi występującymi w algorytmie RC4 (pkt 2.4.1, alg. 2.1). Natomiast wyliczenie wartości wynikowego bajtu jest już bardziej skomplikowane niż w dwóch poprzednich przypadkach. Oprócz – jak w przypadku szyfrów RC4 i VMPC – wyłącznie odwołań do indeksów, w szyfrze RC4+ pojawiają się dodatkowo przesunięcia bitowe, alternatywa wykluczająca oraz stała `0xAA` (zapis szesnastkowy). Ma to na celu utrudnienie skutecznej kryptoanalizy.

Algorytm 2.3: FAZA PRGA SZYFRU RC4+

```
1  i ← 0
2  j ← 0
3  for b ← 1 to L do
4      i ← i + 1
5      j ← j + S[i]
6      zamień (S[i], S[j])
7      t ← S[i] + S[j]
8      t' ← (S[i ≫ 3 ⊕ j ≪ 5] + S[i ≪ 5 ⊕ j ≫ 3]) ⊕ 0xAA
9      t'' ← j + S[j]
10     wynik ← (S[t] + S[t']) ⊕ S[t'']
```

Znane ataki

Podobnie jak w przypadku RC4 przeszukiwanie wyczerpujące wymaga sprawdzenia 2^{2048} kluczy lub 2^{1684} $(256!)$ permutacji początkowych. Dwa ataki na szyfr RC4+ zostały opisane przez Subhadeep Banik i in. [BSK13]. Pierwszy z nich to atak rozróżniający, który wymaga 2^{26} próbek strumieni szyfrujących wygenerowanych przez różne klucze. Jak wspomniano wcześniej, atak tego typu bezpośrednio nie ujawnia nic na temat samego klucza czy szyfrogramu. Drugi atak polegał na różnicowej analizie błędów. I choć w efekcie pozwalał na odnalezienie stanu wewnętrznego szyfru RC4+, wymagał ingerencji w szyfr w formie wstrzyknięcia 2^{17} błędów.

Rozdział 3

Zastosowanie metaheurystyk w kryptoanalizie

W kolejnych podrozdziałach zostanie przedstawiona kryptoanaliza różnego rodzaju szyfrów, od historycznych po współczesne (rys. 2.1), za pomocą algorytmów metaheurystycznych (tab. 1.1). Jako uzupełnienie zostaną także zarysowane przykładowe możliwości użycia algorytmów metaheurystycznych na rzecz kryptografii.

3.1 Metaheurystyki w szyfrach klasycznych

Szyfry przestawieniowe zostały poddane kryptoanalizie z wykorzystaniem różnych algorytmów metaheurystycznych. Russell i in. użyli w tym celu algorytmu optymalizacji mrowiskowej (ACO) i połączyli dwie funkcje oceny: bigramy, które stanowiły element eksploracji oraz ocenę słownikową jako element eksploatacji. System działa najlepiej dla szyfrogramów o małej szerokości oraz dużej wysokości (długie teksty z krótkim kluczem). Dobre rezultaty osiągnięto nawet przy kluczu o długości 40 znaków, przy 5000 iteracji (atak wyczerpujący to $40! \approx 10^{48}$ możliwości do sprawdzenia) [RCS03].

Przeszukiwanie całej przestrzeni rozwiązań jest lepsze w algorytmie genetycznym, lecz zdolności lokalnego przeszukiwania są już gorsze – odwrotnie jest w przypadku symulowanego wyżarzania. Song i in. zaprezentowali algorytm SAGA, czyli połączenie symulowanego wyżarzania (SA) z algorytmem genetycznym (GA), który w założeniach ma łączyć zalety obu algorytmów nadrzędnych [SYWZ08]. Atakowi z szyfrogramem poddano szyfr przestawieniowy o długości klucza do 30 znaków. Osiągnięto wyniki lepsze niż przy użyciu algorytmu genetycznego, przeszukiwania z tabu (TS) i symulowanego wyżarzania

w pracy Dimovskiego i Gligoroskiego [DG03a]. Wyniki częściowo udało się jeszcze poprawić korzystając z tych samych ww. wymienionych algorytmów [Gar09].

Kolejne badania przeprowadzone przez Heydari i in. obejmowały kryptoanalizę przy użyciu algorytmów genetycznych dla kluczy o długości do 25 znaków [HSH13]. Skutecznego ataku kryptoanalitycznego szyfrów przestawieniowych dokonali również Boryczka i Dworak stosując algorytmy ewolucyjne (EA) [BD14b, BD14a, Dwo14].

Lasry i in. jako bazowego algorytmu do kryptoanalizy użyli metody wspinaczkowej (HC). Ich celem było złamanie kolumnowego szyfru przestawieniowego. Znacząco poprawili bazowy algorytm metody wspinaczkowej, stosując specjalistyczne transformacje, wyspecjalizowane funkcje dopasowania, a także dodanie fazy wstępnej, która opiera się na dogłębnej analizie związków między kolumnami szyfrogramu. Proponowana metoda okazała sę bardzo skuteczna w kryptoanalizie szyfrów o długim kluczu, aż do 1 000 znaków. Wymaga również znacznie krótszych szyfrogramów [LKW16, Las18]. Ci sami autorzy również przy użyciu metody wspinaczkowej przeprowadzili atak na historyczny szyfr podstawieniowy ADFGVX [LNKW17, Las18].

Atak na bardzo prosty szyfr podstawieniowy przedstawili Uddin i Youssef. W tym celu użyli optymalizacji stadnej cząsteczek (PSO) i przeprowadzili kryptoanalizę dla tekstów o długości 50–1000 znaków. W pracy przedstawiono porównanie kryptoanalizy opartej na unigramach i bigramach – ta druga daje lepsze rezultaty i pozwala na odzyskanie znaczącej części klucza (ponad 20 z 26 znaków) już dla tekstów o długości 300 znaków [UY06]. Optymalizacji stadnej cząsteczek w kryptoanalizie monoalfabetycznego szyfru podstawieniowego użył również Kielijan [Kie18].

Z kolei Mudgal i in. wykorzystali do kryptoanalizy monoalfabetycznego szyfru podstawieniowego algorytm genetyczny. Najlepsze rezultaty, czyli najszybszą zbieżność do klucza o najwyższej wartości funkcji dopasowania, osiągnięto dla selekcji losowej z elityzmem, krzyżowaniem jednopunktowym oraz mutacją zamiany [MPSJ17]. Również na potrzeby kryptoanalizy monoalfabetycznego szyfru podstawieniowego Luthra i Pal stworzyli integrację operatorów mutacji i krzyżowania powszechnie stosowanych w algorytmach genetycznych z algorytmem świetlika (FA). W ramach eksperymentów zaobserwowano, że algorytm działał lepiej przy dłuższych szyfrogramach. Przy krótszych szyfrogramach potrzebna była większa liczba pokoleń [LP11].

Antal i Eliáš poddali kryptoanalizie monoalfabetyczny szyfr podstawieniowy korzystając z algorytmu genetycznego oraz równoległego algorytmu genetycznego (ang. *Parallel Genetic Algorithm*, PGA). Przeprowadzone badania skupiają się na analizie, jak dobór parametrów wpływa na wskaźnik sukcesu. Na podstawie eksperymentów ustalono, że zwiększenie rozmiaru populacji i liczby iteracji pozwala znaleźć lepsze rozwiązania dla krótszych tekstów. Przy użyciu algorytmu genetycznego udało się uzyskać 80% sukcesów

dla tekstów o długości 500 liter, podczas gdy równoległa wersja algorytmu genetycznego pozwoliła zwiększyć wskaźnik sukcesu do blisko 100% [AE17].

Podobne badania, choć przy wykorzystaniu metody wspinaczkowej zostały przeprowadzone przez Antala w rozprawie doktorskiej [Ant17]. Autor zwraca uwagę na kluczową rolę, jaką odgrywa odpowiednio dobrana funkcja dopasowania. Dla tekstów o długości 200 liter znalezione rozwiązania były zgodne średnio w ponad 90% z rozwiązaniem poszukiwanym. W ramach eksperymentów udało się z powodzeniem poddać kryptoanalizie również szyfr homofoniczny. Z powodzeniem złamano szyfry z nawet 60 homofonami dla bardzo krótkich tekstów (o długości 300 liter).

Bardziej skomplikowaną wersję szyfrów podstawieniowych, a konkretnie szyfr polialfabetyczny, poddali kryptoanalizie Morelli i Walde, do czego zaprzęgnięty został algorytm genetyczny. Badano wiadomości o długościach 250–4000 znaków oraz z kluczami długości 5, 10, 20 i 26 znaków. Niezależnie od długości klucza najlepsze wyniki osiągano dla wiadomości podzielonych na kolumny 100-znakowe, a blisko 100% skuteczność osiągnięto przy 120-znakowych kolumnach. Skuteczność w tym przypadku oznacza liczbę całkowicie odszyfrowanych wiadomości w stosunku do wszystkich prób kryptoanalizy [MW06]. Analizowany szyfr jest równoważny szyfrującej maszynie wirnikowej z jednym wirnikiem.

Bhateja i in. zajęli się kryptoanalizą szyfru Vigenère'a. Algorytm, który wybrali, czyli system kukułczy (CS), okazał się osiągać lepsze rezultaty niż algorytm genetyczny oraz optymalizacja stadna cząsteczek. Dwie ostatnie pozwoliły na odzyskanie całego klucza tylko dla kluczy o małej długości. Z kolei system kukułczy daje możliwość uzyskania powyżej 90% klucza dla kluczy o długości nawet do 25 znaków. Dzieje się tak dlatego, że algorytm genetyczny oraz optymalizacja stadna cząsteczek grzęzną w optimach lokalnych, natomiast system kukułczy znajduje optimum globalne dzięki lotom Lévy'ego. System kukułczy charakteryzuje się nie tylko lepszą dokładnością, ale również szybszą zbieżnością w stosunku do optymalizacji stadnej cząsteczek oraz algorytmu genetycznego [BBCS15].

Lin i Kao przetestowali odporność szyfru Vernama na atak z szyfrogramem z wykorzystaniem algorytmu genetycznego [LK95]. Z kolei Mekhaznia i Menai poddali kryptoanalizie całą gamę szyfrów klasycznych (między innymi Vigenère'a, podstawieniowy, przestawieniowy, afiniczny) zaatakowanych kilkoma wersjami algorytmów mrowiskowych, a porównawczo również algorytmem genetycznym. Osiągnięte rezultaty pokazały przewagę algorytmu optymalizacji mrowiskowej nad algorytmem genetycznym. Autorzy sugerują, że jeszcze lepsze wyniki można osiągnąć korzystając z algorytmu pszczelego lub innych algorytmów inspirowanych inteligencją roju [MM14].

Bagnall poddał kryptoanalizie maszynę szyfrującą z trzema wirnikami. W tym celu użyto algorytmu genetycznego ze współczynnikiem korelacji jako funkcją oceny, selekcją koła ruletki, krzyżowaniem PMX i populacją wielkości 50 osobników. Zdecydowano się na

Tabela 3.1: Lista zastosowań algorytmów metaheurystycznych w kryptoanalizie szyfrów klasycznych

Algorytm	Rodzaj szyfru	Rok	Literatura
HC	przestawieniowy	2016, 2018	[LKW16, Las18]
	podstawieniowy	2017	[Ant17]
	podstawieniowy	2017, 2018	[LNKW17, Las18]
GA	Vernama	1995	[LK95]
	wirnikowa maszyna szyfrująca	1996	[Bag96]
	przestawieniowy	2003	[DG03a]
	polialfabetyczny	2006	[MW06]
	przestawieniowy	2008	[SYWZ08]
	przestawieniowy	2009	[Gar09]
	przestawieniowy	2013	[HSH13]
	przestawieniowy	2014	[MM14]
	podstawieniowy	2014	[MM14]
	afiniczny	2014	[MM14]
	Vigenère'a	2014	[MM14]
	Vigenère'a	2015	[BBCS15]
	podstawieniowy	2017	[MPSJ17]
	podstawieniowy	2017	[AE17]
EA	przestawieniowy	2014	[BD14b, BD14a, Dwo14]
SA	przestawieniowy	2003	[DG03a]
	przestawieniowy	2008	[SYWZ08]
	przestawieniowy	2009	[Gar09]
TS	przestawieniowy	2003	[DG03a]
	przestawieniowy	2009	[Gar09]
ACO	przestawieniowy	2003	[RCS03]
	przestawieniowy	2014	[MM14]
	podstawieniowy	2014	[MM14]
	afiniczny	2014	[MM14]
	Vigenère'a	2014	[MM14]
PSO	podstawieniowy	2006	[UY06]
	Vigenère'a	2015	[BBCS15]
	podstawieniowy	2018	[Kie18]
FA	podstawieniowy	2011	[LP11]
CS	Vigenère'a	2015	[BBCS15]

atak z szyfrogramem. Ponieważ dla tekstów o długości 2704 znaki zdarzały się ustawienia wirników mające lepszą wartość funkcji oceny niż poszukiwana, zdecydowano się na przyjęcie minimalnej długości analizowanego tekstu na poziomie 3380 liter. Dla tekstów tej wielkości przy 15 wykonaniach odnaleziono prawidłowe ustawienie wirnika tylko 4 razy, jednakże i w pozostałych przypadkach część połączeń była prawidłowo odnaleziona, co może być punktem wyjścia do dalszej analizy. Dla tekstów powyżej 4000 znaków algorytm genetyczny odnalazł pozycję ostatniego wirnika za każdym razem. Średnia liczba pokoleń dla różnych tekstów wynosiła pomiędzy 700 a 1500, średni czas od 110 do 240 sekund, a najdłuższy czas deszyfracji to 435 sekund (7,5 minuty). Poszukiwanie pozycji pozostałych dwóch rotorów najłatwiej osiągnąć algorytmem Bauma [Bag96].

Zbiorczą listę algorytmów metaheurystycznych użytych w kryptoanalizie szyfrów klasycznych umieszczono w tab. 3.1. Po sukcesach w kryptografii klasycznej w zaproszonym artykule Clark wzywał do zbadania, czy kryptografia współczesna jest odporna na ataki z wykorzystaniem algorytmów metaheurystycznych [Cla03].

3.2 Metaheurystyki w szyfrach blokowych

Algorytmy kryptograficzne zbudowane na bazie sieci Feistela to między innymi DES (ang. *Data Encryption Standard*) oraz jego uproszczona, „szkolna” wersja SDES (ang. *Simplified DES*). Nalini i Rao poddali kryptoanalizie właśnie tę uproszczoną wersję. Zastosowali w tym celu symulowane wyżarzanie (SA), przeszukiwanie z tabu (TS) oraz porównawczo algorytm genetyczny (GA). Przeprowadzony atak był atakiem z szyfrogramem. W 48 z 50 eksperymentów klucz został odnaleziony w ciągu 5 iteracji przeszukiwania z tabu. Z rozważanych tutaj trzech algorytmów przeszukiwanie z tabu okazało się algorytmem najbardziej efektywnym, zarówno pod względem procentu odnalezionych kluczy, jak i liczby iteracji. Z kolei ataki poprzez symulowane wyżarzanie były najszybsze (pod warunkiem udanej kryptoanalizy). Algorytm genetyczny średnio potrzebował 5 iteracji po 50 osobników. Niemniej jednak osiągał próg 25% przeszukania przestrzeni rozwiązań w stosunku do ataku siłowego. Wraz ze wzrostem liczby analizowanych szyfrogramów, zwiększa się przewaga przeszukiwania z tabu nad symulowanym wyżarzaniem oraz algorytmem genetycznym [NR05].

Następnie ci sami autorzy zajęli się nie tylko wersją „szkolną”, ale również atakiem na trudniejszy algorytm DES, choć nieco uproszczony w stosunku do wersji używanej w praktyce. Rozważana wersja miała 16-bitowy klucz, nie posiadała S-bloków (S-skrzynek) i składała się tylko z 6 rund. Dla porównania wersja przyjęta jako standard przyjmuje na wejściu 56-bitowy klucz i ma 16 rund. W tym przypadku użyto następujących algorytmów: algorytm genetyczny, optymalizację stadną cząsteczek (PSO), przeszukiwanie

z tabu, adaptacyjny algorytm genetyczny (AGA). Dla SDES algorytmy w kolejności od najlepszej to: przeszukiwanie z tabu, optymalizacja stadna cząsteczek, adaptacyjny algorytm genetyczny. Algorytm genetyczny nie podołał w tym przypadku – uzyskano wyniki gorsze niż dla przeszukiwania wyczerpującego. Z kolei podczas kryptoanalizy DES 16-bitowe klucze zostały odzyskane przez wszystkie użyte algorytmy metaheurystyczne. Najlepiej sprawdziły się przeszukiwanie z tabu oraz optymalizacja stadna cząsteczek – były najszybsze w kontekście czasu wykonania. Autorzy planują rozszerzyć atak na DES z 32- oraz 48-bitowym kluczem z większą liczbą rund oraz S-blokami [NR06].

8-rundową wersją DES zajęli się Husein i in. wykorzystując znaną z literatury kryptoanalizę różnicową. W pracy zostały przedstawione dwa warianty: SPCA (ang. *Stored Pair Cryptanalysis*) oraz GPCA (ang. *Generated Pair Cryptanalysis*). Różnią się one tym, że w pierwszym przypadku 1000 par tekstów służących do kryptoanalizy jest wygenerowanych w sposób tradycyjny („różnicowy"), a w drugim w sposób autorski („genetyczny"). W obu przypadkach w kolejnym kroku algorytm genetyczny jest użyty do odnalezienia 64-bitowego wyjścia dla każdego z ośmiu S-bloków. Zatem algorytm genetyczny może być użyty jako algorytm wspomagający powszechnie znaną kryptoanalizę różnicową lub jako algorytm autonomiczny. Zaproponowane podejście pozwala na odnalezienie klucza szybciej niż z użyciem przeszukiwania wyczerpującego oraz standardowej kryptoanalizy różnicowej. Według deklaracji autorów, ataku można użyć również w przypadku innych algorytmów szyfrujących o budowie podobnej do DES [HBH$^+$07].

Podobnie Laskari i in. bazowali na kryptoanalizie różnicowej, której użyli, by złamać 4- i 6-rundową wersję algorytmu szyfrującego DES. Łącznie zbadali siłę siedmiu różnych algorytmów metaheurystycznych: optymalizacji stadnej cząsteczek w dwóch wariantach (globalnym i lokalnym) oraz ewolucji różnicowej (DE) w pięciu wariantach (standardowym oraz czterech modyfikacjach). Kryptoanaliza różnicowa wyznacza 42 bity 56-bitowego klucza. Pozostałe 14 bitów, zamiast przeszukiwaniem wyczerpującym o złożoności 2^{14}, jest wyliczanych przy pomocy algorytmów ewolucyjnych (PSO oraz DE).

Dla wersji 4-rundowej potrzebne było mniej obliczeń funkcji oceny niż dla przeszukiwania wyczerpującego. Przy liczbie par tekstów równej 20 – sukces osiągnięto w 93–100%, średnio 99,3%, gdzie średnia liczba ewaluacji funkcji oceny wynosiła 1309 (dla wszystkich algorytmów). Przy liczbie 50 par tekstów – sukces osiągnięto w 90–100% przypadków, średnio 99,4%, gdzie średnia liczba ewaluacji funkcji oceny to 982 (dla porównania atak wyczerpujący wymaga $2^{14} = 16\,384$ ewaluacji funkcji oceny). Lokalny wariant optymalizacji stadnej cząsteczek miał lepszy wskaźnik sukcesu, choć w przypadkach, gdy oba warianty znajdują klucz, globalna wersja optymalizacji stadnej cząsteczek potrzebowała mniej ewaluacji funkcji oceny. Standardowa wersja ewolucji

różnicowej potrzebowała najmniej ewaluacji funkcji oceny (576) ze wszystkich rozważanych wersji algorytmów ewolucyjnych.

Efektywność dla wersji 6-rundowej silnie zależała od przyjętej funkcji oceny. Optymalizacja stadna cząsteczek w wersji lokalnej okazała się nieskuteczne, optymalizacja stadna cząsteczek w wersji globalnej osiągnęła sukces na poziomie 35%, a warianty ewolucji różnicowej – 55%. Średnio dla wszystkich algorytmów potrzebne było 5600 ewaluacji funkcji oceny. Niższa skuteczność dla tej wersji DES wynika z różnic wyniku kryptoanalizy różnicowej dla 4 i 6 rund, która stanowi wejście dla algorytmów ewolucyjnych rozważanych w tej pracy. Algorytm można rozszerzyć na pozostałe szyfry oparte na sieci Feistela, jeśli są podatne na kryptoanalizę różnicową [LMSV07a, LMSV07b].

Song i in. zaadaptowali atak z wybranym tekstem jawnym – populacja początkowa nie jest losowa, lecz generowana podobnie jak w kryptoanalizie liniowej, co przyspiesza algorytm genetyczny. Przedmiotem ataku ponownie jest szyfr DES, wersja 4-rundowa z pełnym kluczem. Pominięto permutację początkową (IP) oraz końcową (IP^{-1}), co jest standardową procedurą w przypadku takich algorytmów szyfrujących, ponieważ te fazy nie mają znaczenia kryptograficznego (nie zwiększają złożoności czy siły systemu). Jedno wykonanie algorytmu genetycznego odnajduje 4–13 bitów klucza. Są one ustalonymi bitami populacji początkowej kolejnego wykonania algorytmu genetycznego – i tak aż do znalezienia całego klucza. Podczas przeprowadzonych eksperymentów algorytm potrzebował czterech wykonań, by odnaleźć cały 56-bitowy klucz. Autorzy sugerują, że algorytm może zostać rozszerzony na pozostałe szyfry zbudowane na bazie sieci Feistela [SZMW07]. Badania porównawcze z powyższymi z wykorzystaniem algorytmu optymalizacji mrowiskowej (ACO) i optymalizacji stadnej cząsteczek przeprowadzili Khan i in. Stwierdzili oni, że algorytm optymalizacji mrowiskowej wygrywa pod każdym względem z pozostałymi wyżej wymienionymi. Był to atak ze znanym tekstem jawnym i według deklaracji autorów możliwe jest jego rozszerzenie na inne szyfry blokowe, na przykład AES [KAD13].

Inne badania z tego zakresu to kryptoanaliza algorytmu DES w wersjach 6-rundowej i mniejszych z 42- i 49-bitowymi kluczami przy użyciu algorytmów ewolucyjnych (EA) [YSZ08] czy kryptoanaliza wersji „szkolnej”, czyli SDES, z wykorzystaniem algorytmu genetycznego i memetycznego (MA) [Gar10]. Algorytm memetyczny pozwolił na osiągnięcie lepszych wyników – udało się odnaleźć 5–9 bitów 10-bitowego klucza, podczas gdy algorytm genetyczny odnalazł od 4 do 8 bitów. Atak na DES został także wykonany przez McLaughlina [McL08].

Dobrą, przekrojową pracą jest artykuł Selvi i Purusothamana, w którym porównano starsze, popularne algorytmy takie jak algorytm genetyczny, optymalizacja stadna cząsteczek, symulowane wyżarzanie czy przeszukiwanie z tabu z jednymi z nowszych

algorytmów metaheurystycznych, a konkretnie z algorytmem pszczelim (BA), optymalizacją żerowania bakterii (BFO), systemem kukułczym (CS). Kryptoanalizie poddano SDES (pełny, 10-bitowy klucz) oraz DES w wersjach z 16- i 32-bitowym kluczem. Najgorsze wyniki uzyskano dla optymalizacji żerowania bakterii – być może ten algorytm wymaga jeszcze dopracowania. Algorytm genetyczny również nie poradził sobie z analizowanymi szyframi blokowymi. Najwyższy współczynnik sukcesu uzyskano dla optymalizacji stadnej cząsteczek [SP12a].

Szyfr SDES poddali kryptoanalizie również Rajashekarappa i Soyjaudah. Był to atak z szyfrogramem wykonany przy użyciu przeszukiwania z tabu. Wykazali dwukrotną przewagę w szybkości kryptoanalizy nad atakiem siłowym [RS12]. Atak na szyfr SDES przy użyciu optymalizacji stadnej cząsteczek wykonali Dworak i Boryczka [DB15]. Kryptoanalizy tego samego szyfru przy zastosowaniu algorytmu genetycznego i algorytmu memetycznego dokonali Dworak i in. Był to atak z wybranym tekstem jawnym mający na celu ujawnienie klucza szyfrującego [DNBK16]. Dworak i Boryczka dokonali również kryptoanalizy różnicowej szyfru DES ograniczonego do 6 rund korzystając z algorytmu genetycznego oraz symulowanego wyżarzania. Najlepsze wyniki uzyskano dla algorytmu genetycznego wzbogaconego o heurystyczny operator negacji [DB17].

Atak z wybranym tekstem jawnym na pełną wersję szyfru DES przeprowadzili Amic i in. [ASMR16]. Do tego celu zaadaptowali algorytm genetyczny oraz algorytm świetlika (FA). W fazie eksperymentów ten drugi pozwolił uzyskać wyraźnie lepsze rezultaty pod względem wartości funkcji dopasowania osiągniętej w tej samej liczbie iteracji.

Jako alternatywę dla szyfru DES zaproponowano szyfr blokowy FEAL (ang. *Fast Data Encipherment Algorithm*). Początkowo liczba rund była równa 4, później liczbę rund podwojono, a obecnie jest ona zmienna, przy czym zaleca się użycie 32 rund. Szyfr FEAL w oryginalnej wersji, tj. z czterema rundami, poddali kryptoanalizie Dworak i Boryczka [DB16]. W tym celu zastosowali metodę wspinaczkową (HC), algorytm genetyczny oraz dla porównania przeszukiwanie wyczerpujące. Celem było odtworzenie sześciu podkluczy, a w efekcie – odszyfrowanie szyfrogramu. Do algorytmu genetycznego został wprowadzony dodatkowy heurystyczny operator negacji.

Innym szyfrem blokowym o konstrukcji sieci Feistela jest algorytm TEA oraz jego poprawiona, rozszerzona wersja XTEA. Ten pierwszy może przyjmować zmienną liczbę cykli, sugerowana liczba cykli to 32 (64 rundy Feistela). Hu przedstawił atak na wersję 4- i 5-cyklową algorytmu TEA, a dokonał tego korzystając z algorytmu genetycznego poszerzonego o składnik kwantowy: QGA (ang. *quantum inspired GA*). Poprawiono wyniki prezentowane w innych pracach, bazujące tylko na czystym algorytmie genetycznym, a także przeprowadzono udaną kryptoanalizę przy większej niż w owych pracach liczbie cykli (do

5). Kwantowe chromosomy zawierają więcej informacji niż zwykłe o tej samej długości, co pozwala na bardziej zdywersyfikowaną populację [Hu10].

Oryginalnie zarówno TEA, jak i XTEA na wejściu dostają 128-bitowy klucz. 4-, 7- i 13-rundowy szyfr XTEA z 32-bitowym kluczem to przedmiot badań Itaima i Riff. Użyli oni algorytmu genetycznego podczas procesu znajdowania właściwego podklucza (klucza cyklu) w celu poprawienia wyników kryptoanalizy różnicowej. W przypadku wersji 4-rundowej odnalezienie 29–32 bitów z 32-bitowego klucza wymaga 200 deszyfrowań. Dla wersji 7- i 13-rundowej tych deszyfrowań jest koniecznych 600, uzyskując 27–31 bitów klucza. Dla porównania atak brutalny wymaga 2^{32} deszyfrowań (czyli ponad $4 \cdot 10^{10}$). Przedstawiony hybrydowy algorytm wymaga mniej zasobów obliczeniowych niż tradycyjna kryptoanaliza różnicowa [IR08].

Przykładem algorytmu skonstruowanego jako sieć podstawieniowo-permutacyjna (SPN) jest algorytm szyfrujący AES (ang. *Advanced Encryption Standard*), który składa się w oryginalnej wersji z 10-14 rund. Hospodar i in. poddali taki algorytm kryptoanalizie poprzez analizę poboru mocy. Analizowali wyjście jednego S-bloku korzystając z uczenia maszynowego (ML) [HGDM$^+$11].

Serpent to kolejny algorytm o strukturze sieci podstawieniowo-permutacyjnej. Bafghi i in. poddali analizie wersję 4-, 5-, 6- i 7-rundową (oryginalny to 32-rundowy), z kluczem długości 256 bitów. Użyto czterech rodzajów sztucznych sieci neuronowych (ANN) wspomagająco w kryptoanalizie różnicowej, z czego dwa zawierają symulowane wyżarzanie, pozwalające uniknąć utknięcia w optimum lokalnym. Autorzy wypracowali model reprezentujący problem znajdowania charakterystyki różnicowej jako problem znajdowania najkrótszej ścieżki w grafie skierowanym, co czynią używając sztucznych sieci neuronowych. Według autorów zaproponowany model może zostać rozszerzony na pozostałe szyfry blokowe [BSS08]. Atak na Serpent pojawił się również u McLaughlina [McL08].

Abazari i Sadeghian zajęli się szyfrem PRESENT ograniczonym do 6–8 rund (oryginalnie 31 rund). Przedstawili kryptoanalizę różnicową i algorytm optymalizacji mrowiskowej. Algorytm optymalizacji mrowiskowej znajduje najbardziej prawdopodobne charakterystyki w drugim kroku przedstawionej analizy. Inne szyfry blokowe również podatne są na ten atak [AS12].

Zbiorczą listę algorytmów metaheurystycznych użytych w kryptoanalizie współczesnych szyfrów blokowych umieszczono w tab. 3.2.

Tabela 3.2: Lista zastosowań algorytmów metaheurystycznych w kryptoanalizie szyfrów blokowych

Algorytm	Rodzaj szyfru	Rok	Literatura
ANN	Serpent (4-7 rund)	2008	[BSS08]
HC	FEAL (4 rundy)	2016	[DB16]
GA	SDES	2005	[NR05]
	DES (6 rund)	2006	[NR06]
	DES (8 rund)	2007	[HBH+07]
	DES (4 rundy)	2007	[SZMW07]
	XTEA (4, 7 i 13 rund)	2008	[IR08]
	TEA (4 i 5 cykli)	2010	[Hu10]
	SDES	2010	[Gar10]
	SDES	2012	[SP12a]
	DES (4 rundy)	2013	[KAD13]
	SDES	2016	[DNBK16]
	FEAL (4 rundy)	2016	[DB16]
	DES (6 rund)	2017	[DB17]
EA	DES (6 rund)	2008	[YSZ08]
SA	SDES	2005	[NR05]
	DES	2008	[McL08]
	Serpent (4-7 rund)	2008	[BSS08]
	Serpent	2008	[McL08]
	SDES	2012	[SP12a]
	DES (6 rund)	2017	[DB17]
TS	SDES	2005	[NR05]
	DES (6 rund)	2006	[NR06]
	SDES	2012	[RS12]
	SDES	2012	[SP12a]
MA	SDES	2010	[Gar10]
	SDES	2016	[DNBK16]
ACO	PRESENT (6-8 rund)	2012	[AS12]
	DES (4 rundy)	2013	[KAD13]
AGA	DES (6 rund)	2006	[NR06]
PSO	DES (6 rund)	2006	[NR06]
	DES (4 i 6 rund)	2007	[LMSV07a, LMSV07b]
	SDES	2012	[SP12a]
	DES (4 rundy)	2013	[KAD13]
	SDES	2015	[DB15]
ML	AES (1 S-blok)	2011	[HGDM+11]
DE	DES (4 i 6 rund)	2007	[LMSV07a, LMSV07b]
BFO	SDES	2012	[SP12a]
BA	SDES	2012	[SP12a]
FA	DES	2016	[ASMR16]
CS	SDES	2012	[SP12a]

3.3 Metaheurystyki w szyfrach strumieniowych

Younes przeprowadził atak z wybranym tekstem jawnym na szyfry strumieniowe wykorzystujące liniowe i nieliniowe funkcje zwrotne. Nie sprecyzowano, jakie konkretnie szyfry poddano analizie. Jedyną wzmianką jest to, że nieliniowymi funkcjami są systemy Hadmarda i Bruera. Praktyczne zastosowanie tychże nie jest znane autorce niniejszej rozprawy. W swoich badaniach autorzy użyli algorytmu genetycznego (GA), decydując się na krzyżowanie dwupunktowe (w jednej formie) oraz jedną, prostą mutację. Celem badań była aproksymacja strumienia wyjściowego ww. struktur za pomocą rejestru LFSR. Choć osiągnięto skuteczność 100% w odnalezieniu prawidłowego rejestru, to należy zaznaczyć, że badane rejestry były bardzo krótkie: od 5 do 8 bitów [YAA13, You00].

Podobnej analizy dokonała autorka niniejszej rozprawy, lecz dla znacznie dłuższych rejestrów (do 32 bitów). Analizie z użyciem algorytmu genetycznego poddano rejestry LFSR używane w praktyce, w tym składową szyfru A5/1 używanego obecnie w telefonii komórkowej. Osiągnięto od 68 do 100% zgodności, średnio 84% [PB13].

Badania kontynuowano [PB14a] i w kolejnym etapie celem była aproksymacja przez LFSR strumienia bitów generowanego przez dwa szyfry strumieniowe używane w praktyce (A5/1 i A5/2) oraz generator pseudolosowy (BBS). W pracy rozważano siedem autorskich rodzajów mutacji. Dla generatorów BBS z okresem o długości do 328 w zależności od rodzaju użytej mutacji osiągnięto wyniki z przedziału od 69% (dla mutacji usunięcia zaczepu) do 75% (dla mutacji rozciągającej) aproksymacji atakowanego strumienia bitów. Algorytmy szyfrujące A5/1 i A5/2 składają się odpowiednio z trzech oraz czterech sprzężonych ze sobą rejestrów LFSR. W przypadku tych szyfrów zdecydowano się testować tylko dwa najbardziej obiecujące rodzaje mutacji. Podczas badań przy aproksymacji z wykorzystaniem pojedynczego rejestru LFSR o długości z przedziału 20–30 bitów uzyskano zgodność bitów generowanych przez ów znaleziony rejestr z bitami zadanego wyjścia na poziomie średnim 70%, a dla najlepszych przypadków do 77%. Oznacza to, że korzystając tylko z jednego rejestru LFSR można odszyfrować znaczącą część szyfrogramu.

Rozwinięciem tych eksperymentów było sprawdzenie dwóch autorskich rodzajów krzyżowania [PB14b] podczas analizy szyfrów strumieniowych A5/1 i A5/2. Wyniki były podobne jak poprzednio. Dla wszystkich badań łącznie średnie przystosowanie to niecałe 70%, a najlepsze wartości przystosowania osiągnęły poziom do 76%.

Wnioski z poprzednich badań wykorzystano w ataku na szyfr używany w protokołach SSL i WEP, a mianowicie na szyfr strumieniowy RC4 [PB15]. W pracy rozpatrywano rejestry LFSR o długości do 30 bitów, które służyły do aproksymowania strumienia generowanego przez atakowany szyfr. Podobnie jak w ataku na wcześniejsze szyfry zbadano dwa rodzaje krzyżowania oraz dwa rodzaje mutacji. Sprawdzono również wariant bez

mutacji – w tym przypadku wyniki okazały się o 3% gorsze, więc można wnioskować, że mutacja ma wpływ na poprawę osiąganych rezultatów. Średnio osiągnięto zgodność aproksymowanego strumienia na poziomie 68%, należy jednak zaznaczyć, że są tu uwzględnione również gorsze wyniki przy braku mutacji. W najlepszym przypadku osiągnięto 80% zgodności strumienia generowanego przez szyfr RC4

Kryptoanalizy szyfrów strumieniowych używając algorytmu genetycznego dokonał również Bhateja. W artykule niestety brakuje szczegółów. Atak przeprowadzono między innymi na szyfry oparte na rejestrach LFSR, jednak nie jest sprecyzowane, o jakie szyfry chodzi. Do badań użyto rejestrów o długości od 13 do 23 bitów. Prawidłowo odnaleziono od 3 do 50% bitów klucza [Bha14].

Ferriman zbadał szyfr RC4, próbując określić początkowy stan rejestru S (więcej nt. budowy samego szyfru w punkcie 2.4.1). W praktyce ustalenie początkowego stanu tego rejestru równoznaczne jest z możliwością odczytania całej dalszej korespondencji, bez znajomości klucza. Zbadano trzy algorytmy: algorytm genetyczny, optymalizację stadną cząsteczek (PSO) oraz symulowane wyżarzanie (SA). Osobnikiem jest pewna permutacja 256 elementów (rejestr S), w związku z czym autorzy zdecydowali się skorzystać z operatorów znanych z rozwiązywania algorytmami genetycznymi problemu komiwojażera. Są to: krzyżowanie z częściowym odwzorowaniem (ang. *Partial Mapped Crossover*, PMX) oraz krzyżowanie z rekombinacją krawędzi (ang. *Edge Recombination*, ER), a także mutacja przez zamianę miejscami (ang. *swap mutation*, SwM) i mutacja przez inwersję (ang. *inversion mutation*, InvM). W badaniach wykorzystano adaptacyjne sterowanie mutacją (pobudzenie eksploracji poprzez zwiększenie poziomu mutacji, gdy w populacji następuje stagnacja; zmniejszenie poziomu mutacji w przypadku przeciwnym). Zdecydowano się na selekcję turniejową z elitaryzmem (najlepszy osobnik ma gwarancję przejścia do następnej generacji). Algorytm genetyczny pozwolił na dopasowanie 10–25% początkowej sekwencji strumienia szyfrującego. Autor szacuje, że możliwe jest odnalezienie prawidłowej permutacji w ok. 2^{122} generacji ($\approx 3,8 \cdot 10^{36}$). Najlepsze wyniki osiągnięto z krzyżowaniem PMX oraz mutacją SwM. Porównawczo wykonano również testowanie optymalizacją stadną cząsteczek oraz symulowanym wyżarzaniem. Wyniki uzyskane przez oba te algorytmy okazały się dużo gorsze niż wyniki uzyskane przez algorytm genetyczny. Optymalizacja stadna cząsteczek nie była zdolna do tak sprawnego przeszukiwania przestrzeni rozwiązań, a symulowane wyżarzanie okazało się bardzo powolne [Fer13, FO14].

Zbiorczą listę algorytmów metaheurystycznych użytych w kryptoanalizie współczesnych szyfrów strumieniowych umieszczono w tab. 3.3.

Tabela 3.3: Lista zastosowań algorytmów metaheurystycznych w kryptoanalizie szyfrów strumieniowych

Algorytm	Rodzaj szyfru	Rok	Literatura
GA	systemy Hadmarda i Bruera	2000, 2013	[You00, YAA13]
	A5/1, A5/2	2013, 2014	[PB13, PB14a, PB14b]
	RC4	2013, 2014	[Fer13, FO14]
	oparte na LFSR	2014	[Bha14]
	RC4	2015	[PB15]
SA	RC4	2013	[Fer13]
PSO	RC4	2013	[Fer13]

3.4 Metaheurystyki w szyfrach asymetrycznych

W pracach [GS07, KMP97, Spi93, YS98] do analizy szyfru opartego na problemie plecakowym wykorzystano algorytm genetyczny (GA). Bazą był algorytm wprowadzony przez Spillmana, który dla zbioru 15-elementowego znajduje prawidłowe rozwiązanie po przeszukaniu średnio 2% przestrzeni rozwiązań [Spi93]. Garg i Shastri sugerują, że nie jest to wynik najbardziej optymalny z możliwych, toteż podjęli się modyfikacji algorytmu i lepszego dostrojenia parametrów. Dla zbioru 8-elementowego średnio po 115 generacjach otrzymali prawidłowe wyniki ze średnim przeszukaniem przestrzeni rozwiązań poniżej 50% [GS07]. Podobnego zadania podjęli się autorzy wcześniejszej pracy, Yaseen i Sahasrabuddhe. Znalezienie właściwego klucza wymaga przeszukania średnio 0,09–4% przestrzeni rozwiązań dla problemów plecakowych o rozmiarze do 25. Autorzy taką poprawę wyników zawdzięczają dwóm wprowadzonym funkcjom: *findsol* oraz *checkpairs* [YS98]. Temat był kontynuowany przez Ramani i Balasubramanian [RB11]. Hybrydowe rozwiązanie łączące algorytm genetyczny z metodą wspinaczkową (HC) zostało przedstawione w pracach Matoušek [Mat03] oraz Muthuregunathan i in. [MVR09]. Pojawiły się również badania, w których do kryptoanalizy algorytmu szyfrującego opartego na problemie plecakowym użyto ewolucji różnicowej (DE) [SPM$^+$11] czy algorytmu świetlika (FA) [PSM$^+$11].

Atak kanałem bocznym na RSA, wykorzystując algorytm genetyczny, opisali Ali i Al-salami. Przeprowadzili oni atak z pomiarem czasu szyfrowania w celu odnalezienia użytego 64-bitowego klucza. Otrzymane wyniki doprowadziły do ujawnienia prawidłowo 96% kluczy, używając maksymalnie 1500 wiadomości oraz 100 osobników w populacji. Autorzy sugerują możliwość rozszerzenia na systemy o podobnej podstawie działania, jak na przykład standard podpisu cyfrowego, *Digital Signature Standard* (DSS) czy *Digital Signature Algorithm* (DSA) [AAs04].

Próby kryptoanalizy RSA podjął się również Laskari i in., z użyciem sztucznej sieci neuronowej (ANN). Jak się okazuje sieci są nie tylko w stanie dostosować się do danych

uczących, ale również uzyskują bardzo dobre wyniki w odniesieniu do zestawów testowych [LMSV07b].

W tej samej pracy [LMSV07b] przedstawiono atak na DLP. Ponownie użyto sieci neuronowej. Dla małych liczb pierwszych ($p < 200$) sztuczna sieć neuronowa odniosła sukces dochodzący do 100%. Dla większych liczb uczenie sieci neuronowej było trudniejsze. Dla bardzo dużych liczb pierwszych wydajność sieci była niewielka. W pracy rozważono pięć różnych modeli trenujących. Nie znaleziono jednego dominującego modelu. Dla małych liczb pierwszych jeden model pozwalał osiągnąć najlepsze wyniki, dla dużych liczb pierwszych – inny.

Ponownie Laskari i in. przeprowadzili atak na system kryptograficzny oparty na krzywych eliptycznych, tym razem dodatkowo wzbogacając go o ewolucję różnicową. Zbadali system kryptograficzny oparty na krzywych eliptycznych nad ciałem skończonym liczby pierwszej o długości 14, 20 i 32 bitów. Przeanalizowano trzy rodzaje algorytmów uczących i, co warte zauważenia, jeden z nich bazował na ewolucji różnicowej. Przedstawiono następujące wyniki: wyliczenie najmniej znaczącego bitu (*least significant bit*, LSB) logarytmu dyskretnego nad krzywą eliptyczną ze średnią dokładnością 57% na zbiorze testowym (czyli wynik niewiele lepszy niż w przypadku wyboru losowego) oraz 90% na zbiorze treningowym (autorzy sugerują możliwość kompresji zbioru w ten sposób) [LMS$^+$07, LMSV07b].

Tabela 3.4: Lista zastosowań algorytmów metaheurystycznych w kryptoanalizie szyfrów asymetrycznych

Algorytm	Rodzaj szyfru	Rok	Literatura
ANN	RSA	2007	[LMSV07b]
	DLP	2007	[LMSV07b]
	ECC	2007	[LMS$^+$07, LMSV07b]
HC	plecakowy	2003	[Mat03]
	plecakowy	2009	[MVR09]
GA	plecakowy	1993	[Spi93]
	plecakowy	1997	[KMP97]
	plecakowy	1998	[YS98]
	plecakowy	2003	[Mat03]
	RSA	2004	[AAs04]
	plecakowy	2007	[GS07]
	plecakowy	2009	[MVR09]
	plecakowy	2011	[RB11]
DE	ECC	2007	[LMS$^+$07, LMSV07b]
	plecakowy	2011	[SPM$^+$11]
FA	plecakowy	2011	[PSM$^+$11]

Zbiorczą listę algorytmów metaheurystycznych użytych w kryptoanalizie współczesnych szyfrów asymetrycznych umieszczono w tab. 3.4.

3.5 Inne zastosowania metaheurystyk w kryptologii

Algorytmy metaheurystyczne są wykorzystywane nie tylko do łamania szyfrów, lecz również do projektowania bezpieczniejszych systemów kryptograficznych. W literaturze pojawiły się propozycje szyfrów opartych na algorytmach metaheurystycznych, wykorzystano też te algorytmy pomocniczo przy generowaniu struktur używanych w kryptografii.

Przykładowo, algorytmy genetyczne (GA) stały się bazą do stworzenia kilku szyfrów. Jest to o tyle interesujące, że podstawa działania algorytmów metaheurystycznych jest wręcz zaprzeczeniem podstaw, które musi spełniać system kryptograficzny. Algorytmy metaheurystyczne często opierają się na pewnej losowości, a każde wywołanie takiego algorytmu może generować trochę inne wyniki. Z kolei od systemu kryptograficznego wymaga się, aby był w pełni deterministyczny – to znaczy, że po zaszyfrowaniu tekstu jawnego, a następnie jego odszyfrowaniu, wymagane jest otrzymanie dokładnie tej wiadomości, która została zaszyfrowana. Co więcej – wynik musi być przewidywalny i powtarzalny. Nalini i Rao zmodyfikowali algorytm genetyczny tak, aby operatory krzyżowania i mutacji były deterministyczne. Taka wersja algorytmu służy do wygenerowania liczb pseudolosowych, które następnie są wykorzystywane do zaszyfrowania oraz odszyfrowania wiadomości. Wadą tego systemu jest to, że szyfrogram jest dwa razy dłuższy niż tekst jawny [NR04]. Almarimi i in. zaproponowali włączenie procesu krzyżowania znanego z algorytmu genetycznego bezpośrednio do etapu szyfrowania i odszyfrowywania danych. W swoich badaniach eksperymentalnych autorzy wykazali przepustowość wystarczająco dobrą dla rzeczywistych zastosowań [AKAE08].

Szyfr strumieniowy, bazujący na pseudolosowej sekwencji generowanej przez algorytm genetyczny, zaproponowali Saveetha i Arumugam [SA13a]. Aby stworzyć na przykład szyfr strumieniowy można również użyć algorytmów metaheurystycznych zaadaptowanych do programowania, to znaczy algorytmów programowania genetycznego (inspirowane algorytmem genetycznym), programowania symulowanego wyżarzania (inspirowane symulowanym wyżarzaniem, SA) oraz adaptacyjnego programowania genetycznego (inspirowane adaptacyjnym algorytmem genetycznym, AGA) [Awa11]. Szaban i in. przedstawili mechanizm poszukiwania oparty na algorytmie genetycznym, który pozwolił wyselekcjonować zbiór reguł dla automatu komórkowego (CA) do stworzenia dobrej jakości generatora pseudolosowego [SSB06]. Wcześniejszą pracą z zakresu użycia automatu komórkowego do konstrukcji szyfru strumieniowego była praca Bouvry i in.

[BKS05]. Z kolei Sarkar i Mandal [SM14] przedstawili możliwość generowania strumienia szyfrującego z użyciem algorytmu optymalizacji mrowiskowej (ACO).

W artykule Jhajharia i in. pojawił się hybrydowy algorytm łączący optymalizację stadną cząsteczek (PSO) z algorytmem genetycznym, stanowiący bazę szyfru asymetrycznego. Autorzy przeprowadzili szereg testów statystycznych, by wykazać dobrą jakość zaproponowanego rozwiązania [JMB13]. Inny algorytm z wykorzystaniem klucza publicznego oparty jedynie na algorytmie genetycznym zaproponowali Naik i in. [NN14]. Więcej przykładów można znaleźć w [AM14, AAE14, BD13, MNS$^+$13, NPK13].

Kolejnym zastosowaniem może być projektowanie silnie nieliniowych, zrównoważonych funkcji boolowskich przydatnych w kryptografii. Takie funkcje wykorzystywane są do budowy szyfrów zarówno strumieniowych, jak i blokowych [Car10]. W tych pierwszych służą między innymi do zaburzenia liniowości wyjścia rejestrów LFSR, w tych drugich używane są do projektowania na przykład S-bloków. Jakość owych funkcji wpływa na odporność całego algorytmu szyfrującego na różnego rodzaju ataki. Millan i in. zaproponowali użycie algorytmu genetycznego w celu znalezienia zrównoważonych funkcji boolowskich, które będą spełniały kryterium odporności korelacyjnej oraz ścisłe kryterium lawinowości [MCD98]. Taka niedeterministyczna metoda stanowi ciekawą alternatywę dla technik systematycznego budowania kryptograficznie silnych funkcji boolowskich. Podobne podejście opisano również w [AVR14, DG03b, Naj10]. W tym samym celu możliwe jest również użycie także algorytmów ewolucyjnych (EA) [GY12] oraz symulowanego wyżarzania, algorytmu memetycznego (MA) czy algorytmu optymalizacji mrowiskowej [McL12]. W późniejszej pracy Millan i in. użyli algorytmu genetycznego do projektowania S-bloków o wysokiej nieliniowości oraz niskiej autokorelacji [MBC$^+$99]. Projektowaniem S-bloków, lecz przy użyciu automatu komórkowego, zajmowali się również Szaban i Seredyński [SS11b, SS11a].

W szeregu prac Kotlarz i Kotulski zaproponowali użycie sztucznej sieci neuronowej (ANN) do projektowania odpornych na kryptoanalizę S-bloków [KK05b], a następnie opracowali neuronowy układ szyfrujący [KK05a, KK06, KK07a, KK07b, KK08]. Ów układ udostępnia możliwość realizacji różnych szyfrów symetrycznych blokowych opartych na operacjach permutacji oraz podstawiania, czyli dwóch podstawowych przekształceń kryptograficznych. Zmiana używanego algorytmu szyfrującego, zgodnie z założeniami, ma następować poprzez uczenie sieci neuronowej. Jest to ciekawa alternatywa dla sprzętowych układów programowalnych. Podsumowaniem tych rozważań jest bogata w szczegóły rozprawa doktorska [Kot08].

W literaturze można również odnaleźć zastosowanie algorytmów metaheurystycznych do generowania kluczy używanych w znanych szyfrach – algorytmy genetyczne przykładowo dla AES i Blowfish [KB12, RK11, SA13b, Sai14], algorytm optymalizacji

Tabela 3.5: Lista zastosowań algorytmów metaheurystycznych w kryptografii

Algorytm	Zastosowanie	Rok	Literatura
ANN	projektowanie S-bloków	2005-2008	[KK05a, KK06] [KK07a, KK07b, KK08]
CA	szyfr strumieniowy	2004	[BKS05]
	generator pseudolosowy	2006	[SSB06]
	projektowanie S-bloków	2011	[SS11b, SS11a]
GA	projektowanie funkcji boolowskich	1998	[MCD98]
	projektowanie S-bloków	1999	[MBC+99]
	projektowanie funkcji boolowskich	2003	[DG03b]
	system kryptograficzny	2004	[NR04]
	generator pseudolosowy	2006	[SSB06]
	system kryptograficzny	2008	[AKAE08]
	projektowanie funkcji boolowskich	2010	[Naj10]
	system kryptograficzny	2011	[Awa11]
	generowanie kluczy	2011	[RK11]
	generowanie kluczy	2012	[KB12]
	generowanie kluczy	2013	[SA13b]
	system kryptograficzny	2013	[SA13a]
	system kryptograficzny	2013	[BD13]
	system kryptograficzny	2013	[MNS+13]
	system kryptograficzny	2013	[NPK13]
	system kryptograficzny	2013	[JMB13]
	system kryptograficzny	2014	[NN14]
	system kryptograficzny	2014	[AM14]
	system kryptograficzny	2014	[AAE14]
	projektowanie funkcji boolowskich	2014	[AVR14]
	generowanie kluczy	2014	[Sai14]
EA	projektowanie funkcji boolowskich	2012	[GY12]
SA	system kryptograficzny	2011	[Awa11]
	projektowanie funkcji boolowskich	2012	[McL12]
MA	projektowanie funkcji boolowskich	2012	[McL12]
ACO	generowanie kluczy	2008	[SP08]
	generowanie kluczy	2012	[SP12b]
	projektowanie funkcji boolowskich	2012	[McL12]
	system kryptograficzny	2014	[SM14]
AGA	system kryptograficzny	2011	[Awa11]
PSO	generowanie kluczy	2009	[SVP09]
	system kryptograficzny	2013	[JMB13]

mrowiskowej dla sieci komórkowych [SP08] lub innych szyfrów strumieniowych [SP12b] czy optymalizacja stadna cząsteczek również dla szyfrów strumieniowych [SVP09]. Prace te nie będą szczegółowo omawiane, ponieważ jest to poza zakresem niniejszej rozprawy.

Zbiorczą listę algorytmów metaheurystycznych użytych w kryptografii umieszczono w tab. 3.5.

3.6 Podsumowanie

W poprzednich podrozdziałach zaprezentowano rozwiązywanie różnych kryptologicznych problemów przy użyciu algorytmów metaheurystycznych. W tab. 3.6 podsumowano zastosowanie algorytmów metaheurystycznych do zagadnień wymienionych w niniejszym rozdziale.

Tabela 3.6: Zestawienie algorytmów metaheurystycznych zastosowanych w kryptoanalizie i kryptografii

Zastosowane algorytmy	Kryptoanaliza szyfrów:					Kryptografia
	przestawieniowych	podstawieniowych	blokowych	strumieniowych	asymetrycznych	
ANN			✓		✓	✓
HC	✓	✓			✓	
GA	✓	✓	✓	✓	✓	✓
SA	✓		✓	✓		✓
TS	✓		✓			
MA			✓			
ACO	✓	✓	✓			✓
AGA			✓			✓
PSO		✓	✓	✓		✓
ML			✓			
DE			✓		✓	
BFO			✓			
BA			✓			
FA			✓		✓	
CS		✓	✓			

Jak widać, stosunkowo najmniej zbadaną dziedziną jest wykorzystanie algorytmów metaheurystycznych w kryptoanalizie szyfrów strumieniowych. Była to jedna z motywacji do przeprowadzenia na tego typu szyfry ataku posługującego się przeszukiwaniem z tabu oraz algorytmami optymalizacji mrowiskowej. Szczegółowy opis tych metod został omówiony w rozdz. 4 i 5, natomiast badania oraz uzyskane rezultaty znajdują się w rozdz. 6.

Rozdział 4

Algorytm przeszukiwania z tabu w kryptoanalizie

Na bazie przeszukiwania z tabu opracowano autorski algorytm kryptoanalizy wybranych szyfrów strumieniowych, nazwany kryptoanalizą z przeszukiwaniem z tabu. Proponowana strategia kryptoanalizy to atak z tekstem jawnym, mający na celu ujawnienie stanu wewnętrznego algorytmu szyfrującego. Metoda ta zostanie szczegółowo omówiona w niniejszym rozdziale.

Niezależnie od wybranego algorytmu metaheurystycznego są trzy elementy wspólne dla wszystkich algorytmów:

1. reprezentacja rozwiązania,
2. cel,
3. funkcja dopasowania (oceny, przystosowania).

Reprezentacja jest sposobem zakodowania każdego możliwego rozwiązania, na których są wykonywane wszelkie działania wynikające z danego algorytmu. Cel wyraża to, co ma zostać osiągnięte. Funkcja dopasowania określa jakość danego rozwiązania w sposób bezwzględny (gdy znana jest wartość optimum globalnego) lub względny (pozwala na porównanie względnej jakości dwóch rozpatrywanych rozwiązań). Prawidłowe określenie powyższych elementów w konkretnym zastosowaniu jest podstawą, aby dana metaheurystyka miała szansę dać dobre wyniki.

W przedstawionym w rozprawie problemie dopuszczalnym rozwiązaniem jest każda permutacja liczb naturalnych od 0 do zadanego zakresu, w zależności od wielkości analizowanego szyfru ($|S|$). Dla analizowanych w rozprawie algorytmów szyfrujących permutacja ta będzie określać stan wewnętrzny na początku komunikacji, czyli po

zakończeniu fazy KSA, a przed rozpoczęciem PRGA. Skrócony zapis permutacji $(a_0, a_1, a_2, a_3, ..., a_{|S|-1})$ będzie oznaczać odwzorowanie elementu 0 w a_0, elementu 1 w a_1 i tak dalej:

$$(a_0, a_1, a_2, a_3, ..., a_{|S|-1}) = \begin{pmatrix} 0 & 1 & 2 & 3 & ... & |S|-1 \\ a_0 & a_1 & a_2 & a_3 & ... & a_{|S|-1} \end{pmatrix}. \tag{4.1}$$

Zatem permutacja identycznościowa, w której każdy element jest odwzorowany w siebie samego, będzie zapisywana jako:

$$() = (0, 1, 2, 3, ..., |S|-1) = \begin{pmatrix} 0 & 1 & 2 & 3 & ... & |S|-1 \\ 0 & 1 & 2 & 3 & ... & |S|-1 \end{pmatrix}. \tag{4.2}$$

Proponowana metoda kryptoanalizy z przeszukiwaniem z tabu została przedstawiona jako alg. 4.1, gdzie:

TABU – lista tabu,
r – bieżące rozwiązanie,
$R(r)$ – otoczenie bieżącego rozwiązania,
$rBest$ – najlepsza permutacja w całym algorytmie.

Algorytm 4.1: KRYPTOANALIZA Z PRZESZUKIWANIEM Z TABU

Wejście: strumień szyfrujący poddany kryptoanalizie
Wyjście: najlepsza znaleziona permutacja

1 $r \leftarrow$ losowa permutacja
2 $rBest \leftarrow r$
3 TABU $\leftarrow \{\}$
4 $l \leftarrow 0$
5 **while** *nie osiągnięto warunku zatrzymania* **do**
6 $\quad R(r) \leftarrow$ {permutacje powstałe poprzez dopuszczalne zamiany par elementów na r}
7 $\quad r = \arg\max_{i \in R(r)} (f_{fit}(i))$
8 $\quad$ **if** $f_{fit}(r) > f_{fit}(rBest)$ **then**
9 $\quad\quad rBest \leftarrow r$
10 $\quad$ uaktualnij TABU
11 $\quad l \leftarrow l + 1$
12 **return** $rBest$

Działanie algorytmu jest rozpoczynane od wygenerowania losowego stanu wewnętrznego szyfru, a zatem w przypadku rozpatrywanych w niniejszej rozprawie szyfrów: losowej permutacji r (linia 1). Lista tabu (TABU) na początku jest pusta (linia 3).

Następnie identyfikowany jest zbiór $R(r)$ rozwiązań (permutacji) z otoczenia (podrozdz. 4.1) (linia 6). Permutacje, które powstały poprzez zamiany umieszczone na liście tabu, nie są brane pod uwagę. Z otoczenia $R(r)$ wybierane jest najlepsze rozwiązanie, czyli rozwiązanie o najwyższej wartości funkcji dopasowania (podrozdz. 4.2). Jeśli permutacji o najwyższej wartości funkcji dopasowania jest więcej niż jedna, permutacja do kolejnej iteracji jest wybierana spośród nich losowo, z równym prawdopodobieństwem wyboru każdej z nich. Wybrana permutacja (stan początkowy) zostaje nową bieżącą permutacją (linia 7).

Ruch, którego użyto do uzyskania tej wybranej permutacji, zostaje umieszczony na liście tabu i będzie zakazany przez *horyzont* iteracji. Procedura aktualizacji listy tabu składa się z umieszczenia na liście aktualnie wykonanego ruchu (zamiany elementów permutacji) oraz usunięcia najstarszego wpisu, jeśli jest to konieczne (czyli wpisu, który znajduje się na liście więcej niż *horyzont* iteracji) (linia 10).

W kolejnej iteracji ponownie identyfikowane jest otoczenie bieżącego rozwiązania. Ponownie na podstawie funkcji dopasowania wybierane jest najlepsze rozwiązanie z tego otoczenia, przy czym nie brane są pod uwagę ruchy znajdujące się na liście tabu. Analogiczne operacje wykonywane są aż do osiągnięcia warunku zatrzymania (linia 5). Na końcu, jako wynik działania algorytmu (linia 12), zwracana jest permutacja dająca największą wartość funkcji dopasowania (linia 9).

W każdej iteracji jest rozpatrywana tylko jedna permutacja, jednak permutacja zmienia się z iteracji na iterację poprzez zamiany jej wybranych elementów. Początkowa permutacja jest losowa, natomiast nowa permutacja do każdej kolejnej iteracji jest wybierana z otoczenia obecnej permutacji. Jakość permutacji jest oceniana funkcją dopasowania. Im większa wartość tej funkcji, tym lepiej. Nowa permutacja jest wybierana jako najlepsza z otoczenia, z zastrzeżeniem ruchów znajdujących się na liście tabu – te ruchy nie są brane pod uwagę przy analizie otoczenia. Nowo wybierana permutacja nie jest porównywana z obecną pod kątem jakości generowanego strumienia szyfrującego, a zatem możliwe jest zaakceptowanie do kolejnej iteracji permutacji gorszej od obecnej. Mechanizm pamięci dokonanych (i zabronionych) zmian wraz z możliwością akceptacji gorszego rozwiązania ma na celu zmniejszenie ryzyka wpadnięcia w cykl, a tym samym wybicie się z optimum lokalnego i eksplorację innych fragmentów przestrzeni możliwych rozwiązań.

4.1 Otoczenie

Rozwiązaniem z otoczenia permutacji jest również permutacja, niemal identyczna z rozpatrywaną. Permutacje te różnią się od siebie dwoma dowolnymi elementami, które zostały zamienione miejscami. Mając daną permutację: $(a_0, a_1, ..., a_i, ..., a_j, ..., a_{255})$

permutacją z otoczenia jest permutacja postaci $(a_0, a_1, ..., a_j, ..., a_i, ..., a_{255})$ dla każdej pary $i, j \in \langle 0, 255 \rangle$ oraz $i \neq j$. Każda zamiana takich par wygeneruje inną permutację.

W rozprawie zaproponowano i rozpatrzono następujące rodzaje otoczenia:

1. wszystkie pary – otoczeniem jest zbiór zawierający zamiany dwóch dowolnych elementów miejscami (wszystkich możliwych par elementów), w każdej iteracji rozpatrywane są wszystkie takie zamiany;

   - rozmiar potencjalnego otoczenia: $\binom{|S|}{2}$ (liczba sposobów wyboru dwóch elementów z $|S|$, czyli kombinacje bez powtórzeń),
   - rozmiar otoczenia analizowanego w danej iteracji: $\binom{|S|}{2}$ (analizowane jest całe potencjalne otoczenie);

2. połowa par – otoczeniem jest zbiór zawierający zamiany dwóch dowolnych elementów miejscami, w każdej iteracji rozpatrywana jest losowo wybrana połowa takich zamian;

   - rozmiar potencjalnego otoczenia: $\binom{|S|}{2}$ (jak w punkcie 1),
   - rozmiar otoczenia analizowanego w danej iteracji: $\left\lfloor \binom{|S|}{2}/2 \right\rfloor$ (analizowana jest połowa potencjalnego otoczenia);

3. elementy losowo – otoczeniem jest zbiór zawierający po jednej zamianie dla każdego elementu permutacji z innym losowo wybranym elementem, różnym dla każdej iteracji;

   - rozmiar potencjalnego otoczenia: $\binom{|S|}{2}$ (jak w punkcie 1),
   - rozmiar otoczenia analizowanego w danej iteracji: $|S|$ (wybierana jest tylko jedna zamiana dla każdego z $|S|$ elementów);

4. elementy sąsiadująco – otoczeniem jest zbiór zawierający po jednej zamianie dla każdego elementu permutacji z elementem następnym oraz zamiana ostatniego elementu z pierwszym;

   - rozmiar potencjalnego otoczenia: $|S|$ (jest $|S|$ elementów, więc istnieje $|S|$ tego typu zamian),
   - rozmiar otoczenia analizowanego w danej iteracji: $|S|$ (analizowane jest całe potencjalne otoczenie).

Tak zdefiniowane rodzaje otoczenia zawierają co najmniej jedno rozwiązanie i nie obejmują całej przestrzeni rozwiązań. Każde z nich jest zatem prawidłowo zdefiniowanym otoczeniem. Ponadto każde rozwiązanie w całej przestrzeni możliwych rozwiązań jest osiągalne przy użyciu zdefiniowanych powyżej relacji, niezależnie od wyboru osobnika początkowego (uzasadnienie przedstawiono w podrozdz. 4.5).

4.2 Funkcja dopasowania

Dobrze dobrana funkcja dopasowania ma taką właściwość, by przy danej reprezentacji mała zmiana potencjalnego rozwiązania skutkowała niewielką zmianą jej wartości. W rozprawie rozważono i zbadano cztery rodzaje funkcji dopasowania: bajtowe, bajtowe z sąsiadami, ważone i ważone odwrotnie.

W każdym z nich najlepszym możliwym do osiągnięcia wynikiem jest 1, co oznacza, że analizowany strumień bajtów oraz strumień wygenerowany przez algorytm kryptoanalityczny są w 100% zgodne. W tym momencie stan wewnętrzny algorytmu zostaje odnaleziony. Stan wewnętrzny analizowanych szyfrów strumieniowych zmienia się z każdym wygenerowanym bajtem. Z tego powodu wydaje się mało prawdopodobne, by nieprawidłowy stan wewnętrzny pozwolił wygenerować prawidłowy strumień szyfrujący o długości 256 bajtów, a dopiero w dalszej części strumienia generował wartości nieprawidłowe (dokładna analiza prawdopodobieństwa przypadkowej zgodności 256-bajtowego strumienia została umieszczona w podrozdz. 6.4).

Wartość funkcji dopasowania została znormalizowana i może być reprezentowana jako liczba z przedziału $\langle 0, 1\rangle$ lub wartość procentowa z przedziału $\langle 0\%, 100\%\rangle$. Dla przejrzystości, w rozprawie będzie stosowana tylko ta druga konwencja.

Aby zobrazować sposób obliczania funkcji dopasowania przygotowano następujący przykład, który zostanie użyty jako ilustracja obliczania wymienionych i opisanych poniżej funkcji dopasowania. Dla czytelności, strumienie zostały podzielone na bajty, a wartości podane są w postaci szesnastkowej. Obramowaniem zaznaczono bajty, które mają te same wartości na tych samych pozycjach obu strumieni szyfrujących. Podkreśleniem zaznaczono bajty, które mają tę samą wartość występującą na sąsiednich pozycjach w obu strumieniach. Analizowany strumień ma długość $|\kappa| = 16$ B. Niech strumień szyfrujący κ poddany kryptoanalizie będzie następujący:

61-E7-72-3C-7E-4F-51-2A-8B-18-1B-7F-3D-0A-34-C8

Niech strumień szyfrujący κ', wygenerowany w trakcie kryptoanalizy z przeszukiwaniem z tabu, będzie następujący:

61-9E-72-3C-7F-4F-2A-D8-8B-09-E3-CB-4C-E1-EE-34

Poniżej przyjęto następujące oznaczenia:

κ – strumień szyfrujący poddany kryptoanalizie (w bajtach),

κ' – strumień szyfrujący wygenerowany przez aktualnie rozpatrywaną permutację (w bajtach),

$|\kappa|$ – długość strumienia poddanego kryptoanalizie (w bajtach),

κ_m – m-ty bajt strumienia κ

oraz funkcję zgodności dwóch bajtów:

$$B(\kappa'_m, \kappa_m) = \begin{cases} 0, & \kappa'_m \neq \kappa_m, \\ 1, & \kappa'_m = \kappa_m. \end{cases} \tag{4.3}$$

Dopasowanie bajtowe

Funkcja dopasowania bajtowego została określona jako procentowa zgodność bajtów pomiędzy strumieniem analizowanym a strumieniem generowanym przez aktualnie rozpatrywane rozwiązanie (na podstawie [Fer13, FO14]). Funkcja dopasowania to liczba zgodnych bajtów pomiędzy strumieniem wygenerowanym przez rozwiązanie prawidłowe a strumieniem wygenerowanym przez rozwiązanie rozpatrywane:

$$f_{fit}(\kappa') = \frac{\sum_{m=1}^{|\kappa|} B(\kappa'_m, \kappa_m)}{|\kappa|}. \tag{4.4}$$

Wygenerowany strumień prawidłowo pokrył pięć bajtów. Zatem dla tego przypadku wartość funkcji dopasowania bajtowego będzie wynosić:

$$f_{fit}(\kappa') = \frac{5}{16} = 31,3\%. \tag{4.5}$$

Dopasowanie bajtowe z sąsiadami

Funkcja dopasowania bajtowego została zdefiniowana następująco:

$$f_{fit}(\kappa') = \frac{2 \cdot \sum_{m=1}^{|\kappa|} B(\kappa'_m, \kappa_m) + \sum_{m=1}^{|\kappa|-1} B(\kappa'_{m+1}, \kappa_m) + \sum_{m=2}^{|\kappa|} B(\kappa'_{m-1}, \kappa_m)}{2 \cdot |\kappa|}, \tag{4.6}$$

czyli:

- bajt zgodny pomiędzy strumieniem analizowanym a strumieniem wygenerowanym przez rozpatrywane rozwiązanie jest liczony podwójnie,
- dodatkowo bajt zgodny z jednym z sąsiednich bajtów strumienia analizowanego jest liczony pojedynczo.

Suma jest dzielona przez długość analizowanego strumienia pomnożonego przez dwa.

Wygenerowany strumień prawidłowo pokrył pięć bajtów (liczonych podwójnie). Dodatkowo dwa bajty są zgodne z bajtami sąsiednimi w strumieniu analizowanym (liczone pojedynczo). Zatem dla tego przypadku wartość funkcji dopasowania z sąsiadami będzie wynosić:

$$f_{fit}(\kappa') = \frac{2 \cdot 5 + 1 + 1}{2 \cdot 16} = \frac{12}{32} = 37,5\%. \tag{4.7}$$

Dopasowanie ważone

W przypadku dopasowania ważonego zgodność bajtów pomiędzy strumieniem analizowanym a rozpatrywanym jest oceniana tak, że najbardziej istotna jest zgodność pierwszego bajtu, mniej istotna drugiego bajtu i tak dalej, aż po ostatni bajt, którego zgodność jest najmniej istotna:

$$f_{fit}(\kappa') = \frac{\sum_{m=1}^{|\kappa|}(|\kappa| - m + 1) \cdot B(\kappa'_m, \kappa_m)}{\frac{|\kappa|+1}{2} \cdot |\kappa|}. \tag{4.8}$$

Wygenerowany strumień prawidłowo pokrył pięć bajtów, które znajdują się na pozycjach: 1, 3, 4, 6 i 9. Zatem dla tego przypadku wartość funkcji dopasowania będzie wynosić:

$$f_{fit}(\kappa') = \frac{(16-1+1)+(16-3+1)+(16-4+1)+(16-6+1)+(16-9+1)}{\frac{16+1}{2} \cdot 16} = \\ = \frac{16+14+13+11+8}{136} = \frac{62}{136} = 45,6\%. \tag{4.9}$$

Dopasowanie ważone odwrotnie

W przypadku dopasowania ważonego odwrotnie zgodność bajtów pomiędzy strumieniem analizowanym a rozpatrywanym jest oceniana tak, że najbardziej istotna jest zgodność ostatniego bajtu, mniej istotna przedostatniego bajtu i tak dalej, aż do pierwszego bajtu, którego zgodność jest najmniej istotna:

$$f_{fit}(\kappa') = \frac{\sum_{m=1}^{|\kappa|} m \cdot B(\kappa'_m, \kappa_m)}{\frac{|\kappa|+1}{2} \cdot |\kappa|}. \tag{4.10}$$

Wygenerowany strumień prawidłowo pokrył pięć bajtów, które znajdują się na pozycjach: 1, 3, 4, 6 i 9. Zatem dla tego przypadku wartość funkcji dopasowania będzie wynosić:

$$f_{fit}(\kappa') = \frac{1+3+4+6+8}{\frac{16+1}{2} \cdot 16} = \frac{22}{136} = 16,2\%. \tag{4.11}$$

4.3 Złożoność obliczeniowa

Złożoność obliczeniowa każdej iteracji zależy od rozmiaru rozpatrywanego otoczenia i jest z nim powiązana w stosunku liniowym. Dla każdego rozwiązania generowany jest strumień szyfrujący zadanej długości i w pełni porównywany ze strumieniem szyfrującym poddanym analizie (szczegóły funkcji dopasowania w podrozdz. 4.2). Z tego też powodu

złożoność obliczeniowa nie zależy od tego, w jakim stopniu rozpatrywane rozwiązanie jest poprawne.

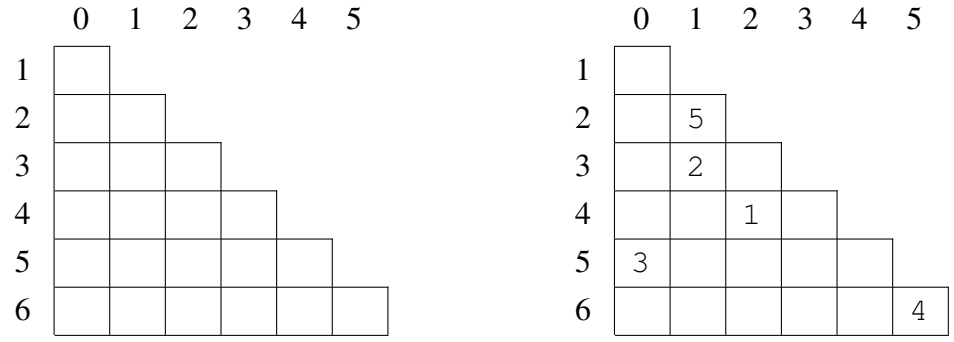
Formalnie czasową złożoność obliczeniową algorytmu przeszukiwania z tabu dla problemu kryptoanalizy można zapisać następująco:

$$T(I, |R(r)|, L) = O(I \cdot |R(r)| \cdot L), \tag{4.12}$$

gdzie:

I – liczba iteracji,
$|R(r)|$ – rozmiar otoczenia,
L – długość strumienia szyfrującego (w bajtach).

Poza aktualnym rozwiązaniem nie ma potrzeby przechowywania w pamięci innych rozwiązań. W pamięci przechowywane są tylko obecne rozwiązanie oraz lista tabu. Lista tabu może być przechowywana jako dwuwymiarowa tablica wszystkich możliwych zamian z wyróżnionymi komórkami zamian zabronionych (rys. 4.1).



(a) Na początku lista tabu jest pusta (b) Lista tabu po kilku iteracjach

Rysunek 4.1: Struktura listy tabu jako dwuwymiarowa tablica wszystkich możliwych zamian; przykład dla permutacji o rozmiarze 7 elementów i horyzontu równego 5

Bardziej oszczędna wersja może polegać na przechowywaniu tylko listy zamian par elementów, na przykład w postaci kolejki FIFO o ustalonej maksymalnej długości, równej horyzontowi (jak w przykładzie z podrozdz. 4.4). W takim przypadku złożoność pamięciowa algorytmu będzie następująca:

$$M(|S|, |\text{TABU}|) = O(|S| + |\text{TABU}|) = O(\max(|S|, |\text{TABU}|)), \tag{4.13}$$

gdzie:

$|S|$ – wielkość permutacji,
$|\text{TABU}|$ – rozmiar listy tabu.

W każdej iteracji sprawdzane są wszystkie rozwiązania z otoczenia, jednak wystarczy zapamiętywać najlepszą uzyskaną wartość funkcji dopasowania oraz numery elementów, które zostały zamienione, by ją uzyskać. Po sprawdzeniu wszystkich otaczających rozwiązań, najlepsze rozwiązanie zostanie łatwo odtworzone na podstawie rozwiązania obecnego oraz zapamiętanych ruchów.

4.4 Przykład

Dla zilustrowania idei działania algorytmu przeszukiwania z tabu rozwiązującego problem kryptoanalizy przedstawiony zostanie prosty przykład wykonania takiej kryptoanalizy. Niech stanem wewnętrznym algorytmu szyfrującego będzie permutacja składająca się z pięciu elementów. Niech rozmiarem listy tabu (horyzont) będzie $|\text{TABU}| = 3$. Niech otoczeniem będzie otoczenie „wszystkie pary".

Wygenerowana losowo została następująca permutacja początkowa (początkowy stan wewnętrzny):
$r = (4, 2, 3, 0, 1)$
Lista tabu na początku jest pusta:
TABU$= \{\}$

Iteracja 1

Istnieje 10 rozwiązań z otoczenia (zamienione elementy zostały podkreślone). Zatem zbiór $R(r)$ składa się z następujących permutacji:
($\underline{2}$, $\underline{4}$, 3, 0, 1)
($\underline{3}$, 2, $\underline{4}$, 0, 1)
($\underline{0}$, 2, 3, $\underline{4}$, 1)
($\underline{1}$, 2, 3, 0, $\underline{4}$)
(4, $\underline{3}$, $\underline{2}$, 0, 1)
(4, $\underline{0}$, 3, $\underline{2}$, 1)
(4, $\underline{1}$, 3, 0, $\underline{2}$)
(4, 2, $\underline{0}$, $\underline{3}$, 1)
(4, 2, $\underline{1}$, 0, $\underline{3}$)
(4, 2, 3, $\underline{1}$, $\underline{0}$)

Dla każdej permutacji (stanu wewnętrznego) możliwe jest obliczenie wartości funkcji dopasowania (zgodnie z przyjętą funkcją dopasowania z podrozdz. 4.2). Założono, że te wartości są następujące:
$(2, 4, 3, 0, 1) : f_{fit} = 9,4\%$
$(3, 2, 4, 0, 1) : f_{fit} = 8,2\%$

(0, 2, 3, 4, 1) : $f_{fit} = 12,1\%$
(1, 2, 3, 0, 4) : $f_{fit} = 10,2\%$
(4, 3, 2, 0, 1) : $f_{fit} = 10,5\%$
(4, 0, 3, 2, 1) : $f_{fit} = 11,7\%$
(4, 1, 3, 0, 2) : $f_{fit} = 7,8\%$
(4, 2, 0, 3, 1) : $\boldsymbol{f_{fit} = 13,3\%}$
(4, 2, 1, 0, 3) : $f_{fit} = 9,8\%$
(4, 2, 3, 1, 0) : $f_{fit} = 7,4\%$
Permutacja z najwyższą wartością f_{fit} została pogrubiona.

Stan wewnętrzny o najwyższej wartości funkcji dopasowania zostaje wybrany do następnej iteracji i staje się nowym stanem bieżącym. Zatem:
r = (4, 2, 0, 3, 1)
Zamiana pomiędzy trzecim a czwartym elementem zostaje umieszczona na liście tabu:
TABU= $\{\{3, 4\}\}$

Iteracja 2

W następnej iteracji rozpatrywane są rozwiązania (permutacje) z otoczenia rozwiązania bieżącego. Są to:
$R(r)$ =
(2, 4, 0, 3, 1) : $f_{fit} = 12,5\%$
(0, 2, 4, 3, 1) : $f_{fit} = 8,6\%$
(3, 2, 0, 4, 1) : $f_{fit} = 9,8\%$
(1, 2, 0, 3, 4) : $f_{fit} = 10,2\%$
(4, 0, 2, 3, 1) : $f_{fit} = 7,4\%$
(4, 3, 0, 2, 1) : $f_{fit} = 12,9\%$
(4, 1, 0, 3, 2) : $f_{fit} = 11,3\%$
~~(4, 2, 3, 0, 1) : $f_{fit} = 10,9\%$~~
(4, 2, 1, 3, 0) : $\boldsymbol{f_{fit} = 13,3\%}$
(4, 2, 0, 1, 3) : $f_{fit} = 10,5\%$
Stan wewnętrzny, który można osiągnąć poprzez zamianę znajdującą się na liście tabu, został przekreślony. Nie jest on brany pod uwagę.

Najlepsze rozwiązanie tej iteracji staje się nowym rozwiązaniem bieżącym:
r = (4, 2, 1, 3, 0)
a zamiana pomiędzy trzecim i piątym elementem zostaje umieszczona na liście tabu:
TABU= $\{\{3, 4\}, \{3, 5\}\}$

Iteracja 3

Zbiór rozwiązań z otoczenia rozwiązania bieżącego wygląda następująco:

$R(r)$ =

(2, 4, 1, 3, 0) : $f_{fit} = 20,3\%$

(1, 2, 4, 3, 0) : $f_{fit} = 18,4\%$

(3, 2, 1, 4, 0) : $f_{fit} = 19,9\%$

(0, 2, 1, 3, 4) : $f_{fit} = 19,1\%$

(4, 1, 2, 3, 0) : $f_{fit} = 16,0\%$

(4, 3, 1, 2, 0) : $f_{fit} = 17,6\%$

(4, 0, 1, 3, 2) : $f_{fit} = 14,5\%$

~~(4, 2, 3, 1, 0) : $f_{fit} = 7,4\%$~~

~~(4, 2, 0, 3, 1) : $f_{fit} = 13,3\%$~~

(4, 2, 1, 0, 3) : $f_{fit} = 9,8\%$

Nowym rozwiązaniem bieżącym, które przechodzi do następnej iteracji, jest:

r = (2, 4, 1, 3, 0)

oraz obecna zawartość listy tabu to:

TABU= $\{\{3,4\},\{3,5\},\{1,2\}\}$

Iteracja 4

Zbiór $R(r)$ składa się z:

~~(4, 2, 1, 3, 0) : $f_{fit} = 13,3\%$~~

(1, 4, 2, 3, 0) : $f_{fit} = 16,0\%$

(3, 4, 1, 2, 0) : $f_{fit} = 13,2\%$

(0, 4, 1, 3, 2) : $f_{fit} = 14,8\%$

(2, 1, 4, 3, 0) : $f_{fit} = 13,6\%$

(2, 3, 1, 4, 0) : $f_{fit} = 13,4\%$

(2, 0, 1, 3, 4) : $f_{fit} = 15,2\%$

~~(2, 4, 3, 1, 0) : $f_{fit} = 18,0\%$~~

~~(2, 4, 0, 3, 1) : $f_{fit} = 12,5\%$~~

(2, 4, 1, 0, 3) : $f_{fit} = 12,9\%$

Jak widać, permutacja o najlepszej wartości dopasowania została osiągnięta przez zamianę, która jest na liście tabu. Z tego powodu ta permutacja nie zostanie wybrana do następnej iteracji. Jako nowa bieżąca permutacja zostanie wybrana permutacja tylko spośród tych osiągniętych przez dozwolone zamiany. Nie tylko nie zostanie wybrana permutacja o najlepszym dopasowaniu, ale również bieżącym rozwiązaniem stanie się permutacja o dopasowaniu gorszym niż poprzednie bieżące rozwiązanie. Zatem nowe

bieżące rozwiązanie to:
r = (1, 4, 2, 3, 0)

Zamiana, która doprowadziła do tej permutacji, zostanie umieszczona na liście tabu. Jednocześnie najstarszy wpis zostanie usunięty, ponieważ długość listy tabu to 3 (co oznacza, że elementy tam umieszczone są przechowywane przez 3 iteracje). Zatem:
TABU$= \{\{3,5\},\{1,2\},\{1,3\}\}$

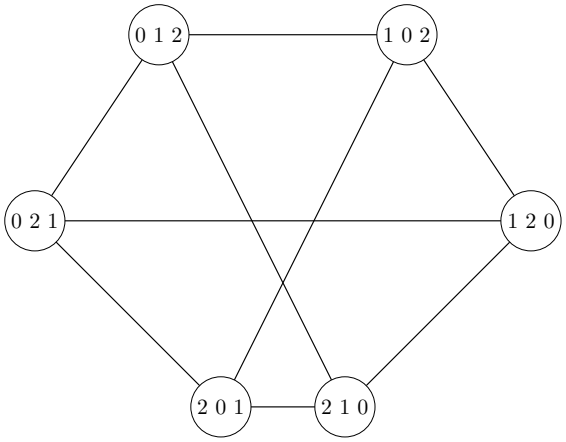
Zakończenie

Dalsze obliczenia przebiegają podobnie, aż do osiągnięcia warunku zatrzymania. Po wyjściu z głównej pętli `while` zostanie zwrócony stan wewnętrzny (permutacja) o najwyższej wartości funkcji dopasowania znalezionej w trakcie działania algorytmu. Niekoniecznie musi być to ostatnia iteracja. W powyższym przykładzie będzie to permutacja: $(2,4,1,3,0)$ z $f_{fit} = 20,3\%$ z iteracji nr 3.

4.5 Generowanie permutacji za pomocą zamian par elementów

Aby wykazać, że otrzymanie poszukiwanej permutacji z permutacji losowo wygenerowanej jest możliwe w skończonej liczbie iteracji, przeprowadzono następujące rozumowanie. Niech wszystkie możliwe permutacje m elementów tworzą graf. Dwa wierzchołki są połączone wtedy i tylko wtedy, gdy permutacje różnią się dokładnie dwoma elementami, które zostały zamienione miejscami. Przykładowy graf dla $m = 3$ przedstawiono na rys. 4.2.

Należy określić najkrótszą ścieżkę pomiędzy najdalszymi wierzchołkami grafu. To znaczy należy odnaleźć najkrótszą ścieżkę prowadzącą pomiędzy najbardziej oddalonymi



Rysunek 4.2: Graf permutacji

od siebie wierzchołkami grafu. Jest to jednoznaczne z obliczeniem, jaka będzie maksymalna liczba zamian par elementów dokonanych na wybranej permutacji, która pozwoli osiągnąć z nich określoną inną permutację.

Niech celem będzie uzyskanie zbioru elementów posortowanych z dowolnej permutacji. Może to zostać osiągnięte z użyciem sortowania przez wybór. Z m elementów należy wybrać największy i zamienić go z ostatnim elementem permutacji (zamiana pary elementów). W tej chwili ostatni element jest już posortowany. W kolejnym kroku należy wybrać element największy z pozostałych, to jest z $m-1$ elementów i zamienić z elementem przedostatnim. Następnie należy wybrać największy z $m-2$ pierwszych elementów i zamienić go z elementem trzecim od końca, i tak dalej aż do posortowania całego zbioru. Aby posortować m elementów jest potrzebnych co najwyżej $m-1$ zamian par elementów.

Analogiczne rozumowanie można przeprowadzić dla przejścia z jednej permutacji do dowolnej innej. Toteż przy m liczbach, aby przejść z jednej permutacji do innej poprzez zamiany par elementów tej permutacji, jest potrzebnych maksymalnie $m-1$ zamian. Zatem w przypadku prezentowanych w niniejszej rozprawie szyfrów potrzebnych jest co najwyżej 255 ($\approx 2^8$) zamian par elementów (iteracji), by z permutacji losowej osiągnąć permutację poszukiwaną (poszukiwany stan wewnętrzny).

Rozdział 5

Algorytmy optymalizacji mrowiskowej w kryptoanalizie

Na podstawie algorytm optymalizacji mrowiskowej opracowano dwa autorskie algorytmy kryptoanalizy szyfrów strumieniowych opartych na permutacji:

1. algorytm mrówki wierzchołkowej (podrozdz. 5.1),
2. algorytm mrowiskowy (podrozdz. 5.2).

W obu przypadkach dopuszczalnym rozwiązaniem jest permutacja liczb z zadanego zakresu. Permutacje są zapisywane w rozprawie zgodnie z wzorami (4.1) i (4.2) (str. 68). Permutacja stanowi stan wewnętrzny analizowanych w rozprawie algorytmów szyfrujących. Celem jest znalezienie permutacji z początku komunikacji, tj. po zakończeniu fazy KSA, a przed rozpoczęciem fazy PRGA. Przedstawiony atak jest atakiem z tekstem jawnym.

5.1 Algorytm mrówki wierzchołkowej w kryptoanalizie

Proponowana metoda kryptoanalizy z algorytmem mrówki wierzchołkowej została przedstawiona jako alg. 5.1, gdzie:

k – mrówka (rozwiązanie w formie permutacji),
A – zbiór mrówek,
$rBest$ – najlepsza mrówka (permutacja) w całym algorytmie.

Każda mrówka (k) to rozwiązanie w formie permutacji. Najpierw zostanie opisana ogólna zasada działania proponowanego algorytmu, a następnie w pkt. 5.1.1-3 zostaną omówione szczegóły dotyczące mapy feromonowej, budowy rozwiązań, liczby mrówek oraz funkcji dopasowania.

Algorytm 5.1: KRYPTOANALIZA Z ALGORYTMEM MRÓWKI WIERZCHOŁKOWEJ

Wejście: strumień szyfrujący poddany kryptoanalizie
Wyjście: najlepsza znaleziona permutacja

```
1  rBest ← NULL
2  zainicjuj mapę feromonową
3  while nie osiągnięto warunku zatrzymania do
4      foreach k ∈ A do
5          zbuduj rozwiązanie (k)
6      foreach k ∈ A do
7          if f_fit(k) > f_fit(rBest) then
8              rBest ← k
9          na podstawie f_fit(k) odłóż feromon na mapie feromonowej
10     wyparuj feromon z mapy feromonowej
11 return rBest
```

Działanie algorytmu rozpoczynane jest od zainicjowania mapy feromonowej domyślną wartością (linia 2). Najlepsze rozwiązanie na początku jest niezdefiniowane (linia 1).

Następnie dla każdej mrówki budowane jest rozwiązanie (linia 5), które jest permutacją. Po ocenie mrówek zgodnie z przyjętą funkcją dopasowania na mapie feromonowej odkładana jest odpowiednia wartość (linia 9). Jeśli w wyniku aktualizacji mapy feromonowej wartość jednego z wierzchołków przekroczy dozwoloną maksymalną wartość feromonu, wartość ta zostanie obniżona do wartości maksymalnej.

W kolejnym kroku następuje wyparowanie feromonu (linia 10). Jeśli w wyniku wyparowania wartość jednego z wierzchołków obniży się poniżej dozwolonej minimalnej wartości feromonu, wartość ta zostanie podwyższona do wartości minimalnej.

W kolejnej iteracji nowe rozwiązania budowane są na podstawie nowej, zaktualizowanej mapy feromonowej. Cała procedura budowania i oceny rozwiązań oraz aktualizacji mapy feromonowej rozpoczyna się od nowa. Algorytm działa iteracyjnie aż do osiągnięcia warunku zatrzymania (linia 3). Na końcu, jako wynik działania algorytmu, zwracana jest permutacja dająca największą wartość funkcji dopasowania (linia 11).

Przedstawiony algorytm został zbudowany na bazie podstawowej wersji algorytmu mrówkowego. Zdecydowano się na feromon cykliczny (ACO), ponieważ dawał on najlepsze wyniki spośród rozpatrywanych opcji [DMC91, Dor92]. Algorytm uzupełniono o znane z systemu mrówkowego MAX-MIN wartości graniczne $\langle \tau_{min}, \tau_{max} \rangle$ dla feromonu, choć w badaniach sprawdzono również wersję bez maksymalnej wartości. Podobnie jak w tej wersji algorytmu ACO, jedną z rozważanych wartości śladu feromonowego domyślnego jest wartość maksymalna śladu feromonowego.

Algorytm mrówkowy wymaga określenia większej liczby parametrów niż przeszukiwanie z tabu. Poniżej opisano dobór tych parametrów.

5.1.1 Feromon

W algorytmie mrówki wierzchołkowej założono, że gęstość śladu feromonowego jest wartością całkowitą dodatnią. Ślad feromonowy $\tau_{b,c}$ w klasycznej wersji algorytmu mrówkowego jest interpretowany jako odwiedzenie wierzchołka c bezpośrednio po wierzchołku b. Jednakże w przedstawionym w rozprawie problemie ślad feromonowy $\tau_{b,c}$ jest interpretowany jako umieszczenie wartości c na b-tym miejscu permutacji. Ponadto, w przeciwieństwie do klasycznego rozwiązania, wprowadzonego wraz z systemem mrówkowym, w proponowanym algorytmie ślad feromonowy nie jest odkładany na krawędziach grafu, lecz w jego wierzchołkach. Krawędzie nie posiadają żadnego kosztu (lub wszystkie krawędzie posiadają koszt jednakowy, co nie ma wpływu na działanie algorytmu). Taka modyfikacja wydaje się być bardziej adekwatna do rozwiązywanego problemu. W tym podejściu nieważny jest koszt tworzonej ścieżki, lecz kolejność poszczególnych wierzchołków oraz ich miejsce występowania na ścieżce.

Ślad feromonowy minimalny: Ślad feromonowy minimalny τ_{min} to wartość, poniżej której nie jest dopuszczalne dalsze wyparowywanie feromonu. W przedstawianym przypadku została ona ustalona na 1. Jest to minimalna wartość liczbowa z założonego zakresu.

Ślad feromonowy maksymalny: Ślad feromonowy maksymalny τ_{max} to wartość, powyżej której nie jest dopuszczalna dalsza kumulacja feromonu. W przedstawianym przypadku przyjęto następujące wartości tego parametru:

- $2|S|$,
- $4|S|$,
- ∞ (brak wartości maksymalnej, feromon może być kumulowany bez żadnych ograniczeń),

gdzie $|S|$ to wielkość permutacji.

Ślad feromony domyślny: Podczas pierwszej iteracji algorytmu mrówkowego konieczne jest odgórne ustalenie domyślnej wartości śladu feromonowego τ_0. W przedstawianym przypadku przyjęto następujące wartości tego parametru zależne od przyjętej wartości śladu maksymalnego:

1. w przypadkach istnienia górnego ograniczenia przyjęto:
    - 1 – jest to najmniejsza możliwa wartość feromonu,

- połowa wartości śladu maksymalnego,
- wartość śladu maksymalnego,

2. w przypadku braku górnego ograniczenia przyjęto:
   - 1 – jest to najmniejsza możliwa wartość feromonu,
   - $|S|$,
   - $|S|^2$.

Aktualizacja mapy feromonowej: Po zbudowaniu rozwiązań przez wszystkie mrówki w danej iteracji, rozwiązania te są oceniane zgodnie z funkcją dopasowania. Następnie wartość funkcji dopasowania uzyskana przez daną mrówkę jest mnożona przez wielkość permutacji i zaokrąglana do najbliższej liczby całkowitej. Tak uzyskana wartość liczbowa stanowi wartość feromonu odkładaną na wszystkich wierzchołkach odwiedzonych przez tę mrówkę.

Wyparowanie feromonu: Z biegiem czasu w środowisku naturalnym następuje swobodne wyparowywanie feromonu. Tak samo w algorytmie mrówki wierzchołkowej po każdej iteracji następuje wyparowanie feromonu. W przeciwieństwie do klasycznej wersji algorytmu, gdzie wyparowanie jest wartością procentową aktualnie skumulowanego feromonu, w niniejszej rozprawie przyjęto, że wyparowana zostaje stała wartość. Przyjęto następujące wartości tego parametru:

- 1 – jest to najmniejsza możliwa do wyparowania jednostka feromonu,
- $|S|/2$,
- $|S|$.

5.1.2 Budowa rozwiązań

W przedstawionym problemie nie ma możliwości probabilistycznego budowania rozwiązań na podstawie zarówno śladu feromonowego, jak i informacji heurystycznej. Ta druga nie jest dostępna w trakcie budowania rozwiązań, dlatego wybór kolejnego kroku przez mrówkę jest opierany wyłącznie na wartości śladu feromonowego. Ponadto nie ma możliwości zbudowania częściowych rozwiązań i ich ewaluacji jeszcze w trakcie budowania rozwiązań.

Na podstawie grafu oraz nałożonego na niego feromonu budowane są rozwiązania w postaci ścieżki zawierającej wszystkie wierzchołki, ale każdy z nich tylko jeden raz. Tak zbudowana ścieżka może być wprost przekształcona na permutację poprzez stworzenie

permutacji z kolejno odwiedzonych wierzchołków. Kolejne wierzchołki do ścieżki są wybierane przez daną mrówkę w sposób losowy, zgodnie z zasadą koła ruletki. Oznacza to, że im większy ślad feromonowy jest odłożony w danym wierzchołku, tym większa jest szansa na jego wybranie. W rozprawie przyjęto trzy rodzaje budowy rozwiązań:

- od początku – każda mrówka jako początek ścieżki ma przydzielony inny wierzchołek; dokładnie jedna mrówka startuje zatem z prawidłowego wierzchołka; pozostałe wierzchołki na ścieżce wybierane są zgodnie z zasadą koła ruletki; ścieżka budowana jest od początku do końca;
- od końca – każda mrówka jako koniec ścieżki ma przydzielony inny wierzchołek; dokładnie jedna mrówka kończy zatem na prawidłowym wierzchołku; pozostałe wierzchołki na ścieżce wybierane są zgodnie z zasadą koła ruletki; ścieżka budowana jest od końca do początku;
- losowo – cała ścieżka jest dla każdej mrówki wybierana zgodnie z zasadą koła ruletki; ścieżka jest budowana od początku do końca; liczba mrówek, które mają pierwszy lub ostatni wierzchołek przydzielony prawidłowo może się wahać w kolejnych iteracjach i być liczbą od 0 do całkowitej liczby mrówek.

Każde rozwiązanie z przestrzeni dopuszczalnych rozwiązań może być wygenerowane przy użyciu powyższej reprezentacji oraz dowolnego z wymienionych sposobów budowy rozwiązań.

5.1.3 Pozostałe parametry

Poniżej umieszczono informacje o parametrach nie omówionych wcześniej.

Liczba mrówek: W algorytmie mrówkowym konieczne jest określenie liczby mrówek w każdej iteracji. W rozprawie przyjęto, że liczba ta jest równa wielkości permutacji ($|A| = |S|$), a tym samym liczbie wierzchołków w grafie.

Funkcja dopasowania: W przedstawianym przypadku użyto tych samych funkcji dopasowania co dla algorytmu przeszukiwania z tabu, a mianowicie: bajtowe, bajtowe z sąsiadami, ważone, ważone odwrotnie. Szczegółowy ich opis znajduje się w podrozdz. 4.2 (str. 71).

5.1.4 Złożoność obliczeniowa

Formalnie czasową złożoność obliczeniową algorytmu mrówki wierzchołkowej dla problemu kryptoanalizy można zapisać następująco:

$$T(I, |S|, |A|, L) = O(I \cdot |A| \cdot L) = O(I \cdot |S| \cdot L), \tag{5.1}$$

gdzie:

I – liczba iteracji,
$|S|$ – wielkość permutacji,
$|A|$ – liczba mrówek,
L – długość strumienia szyfrującego (w bajtach).

W pamięci przechowywane są wszystkie aktualnie rozpatrywane w danej iteracji rozwiązania. Po zakończeniu iteracji są one usuwane, a na ich miejsce powstają kolejne rozwiązania w kolejnej iteracji. Przez cały czas trwania algorytmu w pamięci przechowywana jest mapa feromonowa oraz jedno, najlepsze do tej pory znalezione rozwiązanie.

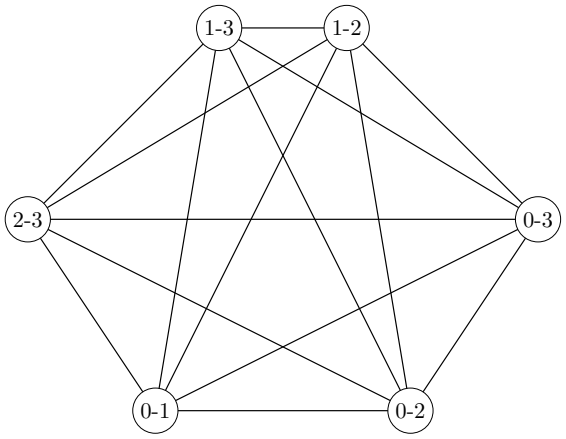
Złożoność pamięciowa algorytmu będzie następująca:

$$\begin{aligned} M(|S|, |A|) = O(|S| \cdot |A| + |S|^2 + |S|) = O(|S| \cdot |S| + |S|^2 + |S|) = \\ = O(|S|^2). \end{aligned} \tag{5.2}$$

5.2 Algorytm mrowiskowy w kryptoanalizie

Przedstawiona wersja algorytmu bazuje na rozwiązaniu problemu komiwojażera. Jest dany graf pełny, co oznacza, że każda para wierzchołków jest połączona krawędzią. Każdy wierzchołek reprezentuje parę zamian pomiędzy elementami permutacji. Mrówka wchodząca do wierzchołka $b = \{i, j\}$ mając daną permutację: $(a_0, a_1, ..., a_i, ..., a_j, ..., a_{255})$ zmienia ją na permutację postaci $(a_0, a_1, ..., a_j, ..., a_i, ..., a_{255})$, gdzie $i, j \in \langle 0, 255 \rangle$ oraz $i \neq j$. Każda zamiana takich par wygeneruje inną permutację. Informacja heurystyczna jest wyliczana na podstawie wartości funkcji dopasowania rozwiązania po dokonaniu takiej zamiany, dla każdego z dostępnych wierzchołków. Każda mrówka posiada własną listę tabu ($tabu$), która przechowuje już odwiedzone wierzchołki. Wierzchołki te są zabronione do odwiedzenia przez daną mrówkę do momentu ukończenia cyklu. Po ukończeniu cyklu lista tabu jest czyszczona i budowanie cyklu jest rozpoczynane na nowo.

Mrówki, które przechowują rozwiązania w formie permutacji, poruszają się w grafie, którego wierzchołki reprezentują pary zamian. Przykładowy graf zamian dla rozmiaru



Rysunek 5.1: Graf zamian

$|S| = 4$ przedstawiono na rys. 5.1. Graf dla takiej 4-elementowej permutacji posiada 6 wierzchołków oraz 15 krawędzi.

Proponowana metoda kryptoanalizy z algorytmem mrowiskowym została przedstawiona jako alg. 5.2, gdzie:

k – mrówka (rozwiązanie w formie permutacji),
A – zbiór mrówek,
q_0 – współczynnik względnej zależności między eksploatacją a eksploracją,
$iBest$ – najlepsza mrówka (permutacja) w iteracji,
$rBest$ – najlepsza mrówka (permutacja) w całym algorytmie.

Każda mrówka (k) to rozwiązanie w formie permutacji. Działanie algorytmu rozpoczynane jest od zainicjowania mapy feromonowej domyślną wartością τ_0 (linia 1). Najlepsze rozwiązanie na początku jest ustalone jako permutacja identycznościowa (linia 2). Taka sama permutacja jest na początku przyporządkowana wszystkim mrówkom (linie 3-4).

Liczba mrówek równa jest liczbie wierzchołków w grafie. Każda mrówka zostaje umieszczona w innym wierzchołku (linia 6). Od razu wykonywana jest zamiana wynikająca z tego wierzchołka (linia 9). Każda tak powstała permutacja jest od razu oceniana (linia 11). Do oceny użyto tych samych funkcji dopasowania, które zostały wprowadzone wraz z algorytmem przeszukiwania z tabu (podrozdz. 4.2, str. 71).

Następnie mrówki będą budować cykl. Każda mrówka będąca rozwiązaniem w postaci permutacji w kolejnym kroku może odwiedzić tylko te wierzchołki, których jeszcze nie odwiedziła w tym cyklu (linia 14). Najpierw dla każdej mrówki losowana jest liczba q (linia 16). Jeśli jej wartość jest mniejsza od ustalonego parametru q_0 (linia 17), nastąpi eksploatacja (linia 18) – mrówka k będąca w wierzchołku b wybierze najlepszy wierzchołek c spośród nieodwiedzonych zgodnie z wzorem (1.9) (str. 27). W przeciwnym razie nastąpi

eksploracja – wybór kolejnego wierzchołka będzie probabilistyczny zgodnie z zasadą koła ruletki (linia 20), a prawdopodobieństwo wyboru zostanie wyliczone z wzoru (1.10) (str. 27).

Po przejściu do kolejnego wierzchołka każda mrówka wykonuje zamianę reprezentowaną przez ten wierzchołek (linia 22), a tak powstałe nowe rozwiązanie jest oceniane (linia 23). Jednocześnie następuje lokalna aktualizacja mapy feromonowej (linia 26). Każda mrówka na krawędzi, którą przeszła, odkłada ślad feromonowy o wartości τ_0 oraz następuje odparowanie feromonu zgodnie z wzorem (1.7) (str. 26).

Po wykonaniu przez mrówki pełnego cyklu, na podstawie wartości funkcji dopasowania jest wybierana najlepsza mrówka danego cyklu (linia 27). Jeśli jest więcej niż jedna mrówka o tej samej największej wartości funkcji dopasowania, rozwiązanie jest wybierane losowo spośród nich. Na tej podstawie następuje globalne uaktualnienie śladu feromonowego (linia 28). Ślad jest nakładany na trasie pokonanej przez najlepszą z mrówek zgodnie z wzorem:

$$\tau_{b,c}(t+1) = \tau_{b,c}(t) + \rho \cdot f_{fit} \tag{5.3}$$

dla każdej krawędzi należącej do trasy najlepszej mrówki.

Wszystkie mrówki otrzymują rozwiązanie najlepszej mrówki jako swoje rozwiązanie bieżące, początkowe w następnym cyklu (linia 30). Następnie budowanie cyklu zaczyna się od nowa.

W kolejnej iteracji nowe rozwiązania budowane są na podstawie nowej, zaktualizowanej mapy feromonowej. Cała procedura budowania cyklu i oceny rozwiązań oraz aktualizacji mapy feromonowej rozpoczyna się od nowa. Algorytm działa iteracyjnie aż do osiągnięcia warunku zatrzymania (linia 5). Na końcu jako wynik działania algorytmu zwracana jest permutacja dająca największą wartość funkcji dopasowania, znaleziona w dowolnym momencie działania algorytmu (linia 31).

Przedstawiony algorytm został zbudowany na bazie podstawowej wersji algorytmu optymalizacji mrowiskowej. Algorytm uzupełniono o znaną z systemu mrowiskowego ACS możliwość optymalizacji jakości kolejnego kroku ze względu na ślad feromonowy oraz informację heurystyczną. Podobnie jak w systemie mrówkowym MAX-MIN na koniec każdej iteracji wybierana jest najlepsza mrówka i tylko ona dokonuje globalnego uaktualnienia śladu feromonowego, a wartość tego uaktualnienia jest zależna od jakości posiadanego przez mrówkę rozwiązania. Ze względu na możliwość oceny rozwiązań w trakcie ich budowania (w trakcie budowy cyklu), najlepsze rozwiązanie jest od razu zapamiętywane.

Algorytm 5.2: KRYPTOANALIZA Z ALGORYTMEM MROWISKOWYM

Wejście: strumień szyfrujący poddany kryptoanalizie, q_0
Wyjście: najlepsza znaleziona permutacja

zainicjuj mapę feromonową
$rBest \leftarrow ()$
foreach $k \in A$ **do**
    $k \leftarrow ()$
while *nie osiągnięto warunku zatrzymania* **do**
    umieść każdą mrówkę w innym wierzchołku
    $tabu_k \leftarrow \varnothing$
    foreach $k \in A$ **do**
        dokonaj zamiany wynikającej z wierzchołka c
        $tabu_k \leftarrow tabu_k + c$
        $f_{fit}(k)$
        if $f_{fit}(k) > f_{fit}(rBest)$ **then**
            $rBest \leftarrow k$
    while *pozostały wierzchołki do odwiedzenia* **do**
        foreach $k \in A$ **do**
            wylosuj liczbę $q \in \langle 0, 1 \rangle$
            if $q \leqslant q_0$ **then**
                wybierz najlepszy wierzchołek $c \notin tabu_k$
            else
                wybierz zgodnie z zasadą koła ruletki następny wierzchołek $c \notin tabu_k$
            $tabu_k \leftarrow tabu_k + c$
            dokonaj zamiany wynikającej z wierzchołka c
            $f_{fit}(k)$
            if $f_{fit}(k) > f_{fit}(rBest)$ **then**
                $rBest \leftarrow k$
        dokonaj lokalnej aktualizacji mapy feromonowej
    $iBest = \arg\max_{k \in A}(f_{fit}(k))$
    dokonaj globalnej aktualizacji mapy feromonowej na podstawie $f_{fit}(iBest)$
    foreach $k \in A$ **do**
        $k \leftarrow iBest$
return $rBest$

5.2.1 Złożoność obliczeniowa

Liczba mrówek jest równa liczbie wierzchołków w grafie, a to z kolei zależy od wielkości rozpatrywanej permutacji S. Zatem:

$$|A| = \frac{|S| - 1 + 1}{2} \cdot (|S| - 1) = \frac{|S|^2 - |S|}{2}. \tag{5.4}$$

Formalnie czasową złożoność obliczeniową algorytmu mrowiskowego dla problemu kryptoanalizy można zapisać następująco:

$$\begin{aligned} T(I, |S|, |A|, L) &= O(I \cdot (|A| + |A| \cdot (\frac{|A| - 1 + 1}{2} \cdot (|A| - 1))) \cdot L) = \\ &= O(I \cdot \frac{|A|^3 - |A|^2 + 2|A|}{2} \cdot L) = O(I \cdot |A|^3 \cdot L) = \\ &= O(I \cdot (\frac{|S|^2 - |S|}{2})^3 \cdot L) = O(I \cdot |S|^6 \cdot L), \end{aligned} \tag{5.5}$$

gdzie:

I – liczba iteracji,
$|S|$ – wielkość permutacji,
$|A|$ – liczba mrówek,
L – długość strumienia szyfrującego (w bajtach).

Złożoność pamięciowa algorytmu (głównie ze względu na wielkość grafu) będzie następująca:

$$M(|S|) = \frac{|S|^2(|S| - 1)^2 - 2|S|(|S| - 1)}{8} = O(|S|^4). \tag{5.6}$$

W pamięci przechowywane są wszystkie aktualnie rozpatrywane w danej iteracji rozwiązania. Przez cały czas trwania algorytmu w pamięci przechowywana jest mapa feromonowa oraz jedno, najlepsze do tej pory znalezione rozwiązanie.

Rozdział 6

Badania eksperymentalne

W niniejszym rozdziale opisano badania eksperymentalne dotyczące proponowanego w rozprawie ataku kryptoanalitycznego. Przedstawiony atak jest atakiem z tekstem jawnym. W podejściu tym zakłada się, że kryptoanalityk dysponuje zarówno tekstem jawnym, jak i odpowiadającym mu szyfrogramem (tekstem zaszyfrowanym). Jest to najbardziej popularna metoda łamania szyfrów [Sch02] i wbrew pozorom jest to prawdopodobna sytuacja w rzeczywistym świecie. Taki atak może mieć miejsce wtedy, gdy przejęta zostanie część wiadomości lub niektóre jej fragmenty można przewidzieć, np. nagłówki plików, stopkę w wiadomości poczty elektronicznej itp.

Łamanie szyfrów ze znanym tekstem jawnym z powodzeniem stosowano podczas II wojny światowej. Meldunki niemieckiej armii miały łatwy do przewidzenia początek, co ułatwiało ich analizę kryptoanalitykom polskim i brytyjskim. Podobnie amerykańscy kryptoanalitycy wspomagali się standardowymi formułami początkowymi i końcowymi w wiadomościach japońskich [Kah04]. W przypadku szyfrów strumieniowych – czyli takich jak rozważane w niniejszej rozprawie – należy na podstawie tekstu jawnego oraz szyfrogramu uzyskać strumień szyfrujący w postaci strumienia bajtów.

W badaniach poddano kryptoanalizie strumień szyfrujący generowany przez trzy różne szyfry strumieniowe. Są to: szyfr RC4, zaimplementowany w protokołach internetowych, i jego odmiana: szyfr RC4+, a także polski szyfr VMPC. Budowa i działanie szyfrów zostały omówione w podrozdz. 2.4.

Wstępne badania kryptoanalityczne przeprowadzono na szyfrach o mniejszych rozmiarach. Dla każdego szyfru wykonano eksperymenty korzystając z kilku różnych kluczy tak, by wyniki można było uznać za uniwersalne, niezależne od specyficznych właściwości klucza. Wartości użytych kluczy zostały podane w postaci szesnastkowej.

Analizie poddano strumień o długości równej wielkości poszukiwanej permutacji, wygenerowany bezpośrednio po procedurze dystrybucji klucza i wektora inicjacyjnego.

Żadne początkowe bajty ze strumienia szyfrującego nie były odrzucane. Celem badań było wygenerowanie takiego strumienia szyfrującego, który jest w pełni zgodny ze strumieniem analizowanym, a co za tym idzie – znalezienie stanu wewnętrznego (permutacji elementów tablicy S) algorytmu szyfrującego z początku komunikacji. Ujawnienie stanu wewnętrznego algorytmu szyfrującego jest jednoznaczne z poznaniem klucza i umożliwia wygenerowanie dalszej części strumienia szyfrującego, a tym samym odszyfrowanie pozostałej części szyfrogramu.

Z każdego wykonania algorytmu przeszukiwania z tabu, algorytmu mrówki wierzchołkowej oraz algorytmu mrowiskowego zwracana jest najlepsza znaleziona permutacja, niezależnie od tego, w której iteracji się pojawiła. Algorytmy użyte do kryptoanalizy są algorytmami probabilistycznymi i każde ich wykonanie może zwrócić inny wynik. Dlatego dla każdego zestawu ustalonych parametrów dla algorytmów wykonano 32 niezależne testy (przyjmuje się, że próba jest duża, gdy liczba elementów z populacji jest większa niż 30). W rozprawie sformułowanie wartość średnia lub uśredniona ($śr.$) odnosi się zawsze do średniej arytmetycznej.

Eksperymenty zostały wykonane za pomocą komputera wyposażonego w procesor Intel Core i7 (3,30 GHz) i 16 GB pamięci RAM DDR 4 oraz z zainstalowanym 64-bitowym systemem operacyjnym Windows 7 Pro. W ramach rozprawy:

1. Zaimplementowano algorytm przeszukiwania z tabu oraz wybrane algorytmy optymalizacji mrowiskowej w języku C# w środowisku programistycznym Visual Studio 2015.

2. Zaimplementowano w języku C# wybrane szyfry strumieniowe: RC4, VMPC i RC4+.

3. Zaimplementowano w języku C# oprogramowanie wspomagające badania.

4. Sprawdzono implementację za pomocą wbudowanego w środowisko Visual Studio narzędzia profilującego (Performance Profiler) w celu poprawienia szybkości działania programu.

5. W języku znaczników XML przygotowano bazę wektorów testowych dla analizowanych szyfrów. Dzięki temu wektory testowe (w tym klucze) są automatycznie wczytywane podczas analizy, a wyniki działania algorytmów szyfrujących są każdorazowo sprawdzane pod kątem poprawności implementacji.

6. W celu wykonania analizy wyników użyto dodatkowo arkusza kalkulacyjnego z pakietu Open Office.

Przez wektor testowy rozumie się trójkę: klucz, wektor inicjacyjny (IV) i strumień szyfrujący wygenerowany przy użyciu tego klucza oraz wektora inicjacyjnego. Obecność

wektora inicjacyjnego jest opcjonalna i zależy od konstrukcji analizowanego algorytmu szyfrującego. Wektory testowe podane w specyfikacji szyfru pozwalają zweryfikować poprawność implementacji. W eksperymentach użyto publicznie dostępnych wektorów testowych dla szyfrów RC4 [SJ11] i VMPC [Żół04], ponieważ tylko w tych przypadkach takie wektory były dostępne. W pozostałych przypadkach w ramach rozprawy wygenerowano autorskie wektory testowe. Wektory testowe starano się wybrać tak, by zapewnić różnorodność analizowanych przypadków, a zatem uwzględniono klucze różnej długości i budowy. Przykładowo, kolejne bajty klucza składają się z kolejnych liczb całkowitych, bajty klucza tworzą powtarzającą się sekwencję, klucz składa się z samych 0 i jednej 1 lub wartość klucza jest losowa. Dla czytelności w opisach eksperymentów podano tylko wartości użytych kluczy i wektorów inicjacyjnych, z pominięciem strumienia szyfrującego.

Badania zostały wykonane osobno dla szyfru RC4 (podrozdz. 6.1), VMPC (podrozdz. 6.2) i RC4+ (podrozdz. 6.3) w następującej kolejności:

- **Eksperymenty 1, 2 i 3**: Dobranie wartości parametrów dla algorytmu przeszukiwania z tabu, algorytmu mrówki wierzchołkowej i algorytmu mrowiskowego na podstawie wyników kryptoanalizy dziesięcioelementowej wersji szyfru. Wyniki tych eksperymentów dały podstawę do odrzucenia algorytmu dającego najgorsze wyniki, a w toku dalszych eksperymentów rozważono tylko dwa pozostałe algorytmy.

- **Eksperymenty 4 i 5**: Sprawdzenie możliwości użycia algorytmu przeszukiwania z tabu i algorytmu mrowiskowego w kryptoanalizie szyfru o liczbie potencjalnych permutacji równej około 2^{44}. Badania wykonano na szesnastoelementowej wersji szyfru. Wyniki tych eksperymentów dały podstawę do wybrania do dalszej części eksperymentów algorytmu pozwalającego osiągnąć najlepsze wyniki.

- **Eksperyment 6**: Sprawdzenie algorytmu przeszukiwania z tabu w kryptoanalizie pełnej wersji szyfru. Wygenerowano i porównano dodatkowo tyle samo rozwiązań losowych.

- **Eksperyment 7**: Porównanie wyników uzyskanych przy użyciu algorytmu przeszukiwania z tabu z wynikami uzyskanymi przy użyciu innych algorytmów metaheurystycznych w ramach podobnych badań opisanych w literaturze (tylko dla szyfru RC4).

Ponadto w podrozdz. 6.4 obliczono prawdopodobieństwo przypadkowej zgodności strumienia szyfrującego. Wyniki porównano z wynikami otrzymanymi w poprzednich eksperymentach.

6.1 Kryptoanaliza szyfru RC4

W poniższych eksperymentach zbadano możliwość użycia zaproponowanych algorytmów w kryptoanalizie najbardziej znanego szyfru strumieniowego RC4.

6.1.1 Dobór wartości parametrów dla algorytmu przeszukiwania z tabu

Celem eksperymentu było sprawdzenie wpływu poszczególnych parametrów algorytmu na jakość otrzymywanych rozwiązań. Do badań wybrano mniejszą wersję szyfru, w której założono, że permutacja przechowywana w tablicy S składa się z dziesięciu liczb z zakresu od 0 do 9. Wersję tę oznaczono jako RC4_10. Zasada działania algorytmu jest taka sama jak w alg. 2.1, z tą różnicą, że wszystkie operacje wykonywane są $\mathrm{mod}\ 10$. Liczba możliwych kluczy dla szyfru RC4 o tej wielkości to 2^{80}, a liczba możliwych początkowych permutacji to $10! = 3\,628\,800 \approx 2^{22}$.

Warunki przeprowadzenia eksperymentu – szczegółowo opisane w dalszej części – były następujące:

- badany algorytm: przeszukiwanie z tabu (alg. 4.1),
- algorytm szyfrujący poddany kryptoanalizie: RC4_10,
- długość analizowanego strumienia szyfrującego: 10 bajtów,
- 10 kluczy (tab. 6.1),
- warunek zatrzymania: sprawdzenie 3 628 880 permutacji lub $f_{fit} = 100\%$,
- losowa inicjacja początkowej permutacji,
- 4 rodzaje funkcji dopasowania (wzory (4.4), (4.6), (4.8), (4.10), str. 72-73),
- 6 rodzajów horyzontu,
- 4 rodzaje otoczenia,
- 2 kryteria aspiracji,
- liczba niezależnych testów dla jednego zestawu parametrów: 32.

Łącznie przeprowadzono 5 120 niezależnych testów ($10 \cdot (4 + 6 + 4 + 2) \cdot 32$, co wynika z liczby różnych przyjętych wartości parametrów).

Przeszukiwanie wszystkich permutacji o rozmiarze 10 daje $10! = 3\,628\,800$ możliwości do sprawdzenia. Dlatego sprawdzenie takiej liczby permutacji ustalono jako

Tabela 6.1: Klucze użyte w kryptoanalizie szyfru RC4_10

Klucz
0x0102030405
0x01020304050607
0x0102030405060708
0x0807060504030201
0x0000000000000001
0x0100040902
0x05060100040902
0x0905000506010004
0x0500080908040905
0x0309020708050909

warunek zatrzymania pracy algorytmu. Przeanalizowanie większej liczby permutacji byłoby bardziej kosztowne niż przeszukiwanie wyczerpujące przestrzeni możliwych permutacji. W przypadku znalezienia permutacji o wartości funkcji dopasowania równej 100% algorytm kończy działanie wcześniej i zwraca znalezione rozwiązanie.

Dobór wartości parametrów algorytmu przeszukiwania z tabu wykonano według następującego schematu: zmieniano wartości ustalonego parametru w zadanym zakresie, przy ustalonych wartościach pozostałych parametrów. Wartość parametru, która umożliwiła uzyskanie najlepszych wyników, była za każdym razem ustalana jako obowiązująca na dalszym etapie eksperymentu. Schemat ten był wykonywany kolejno dla wszystkich wymienionych parametrów. Przyjęto następujące wartości początkowe parametrów:

- funkcja dopasowania: dopasowanie bajtowe (wzór (4.4), str. 72),
- otoczenie: połowa par,
- horyzont $\sqrt{|R(r)|}$ (gdzie $|R(r)|$ to rozmiar otoczenia rozwiązania r),
- kryterium aspiracji: brak.

Jeden test trwał około 0,2 sekundy.

Pierwsze testy wykonano dla funkcji dopasowania. Zaproponowano i rozpatrzono następujące rodzaje funkcji dopasowania: dopasowanie bajtowe (na podstawie [Fer13, FO14]), dopasowanie bajtowe z sąsiadami, dopasowanie ważone, dopasowanie ważone odwrotnie. Na tym etapie wykonano 1 280 testów (4 rodzaje funkcji dopasowania dla 10 kluczy i 32 powtórzeń). Liczbę testów, które zakończyły się sukcesem, tj. znaleziono poszukiwaną permutację przed osiągnięciem maksymalnej liczby sprawdzonych permutacji, umieszczono w tab. 6.2. Najlepsze wyniki (najwięcej odnalezionych permutacji) uzyskano

Tabela 6.2: Wartości funkcji dopasowania dla algorytmu przeszukiwania z tabu (szyfr RC4_10)

Dopasowanie	Liczba sukcesów	
bajtowe	**318**	**99,4%**
bajtowe z sąsiadami	317	99,1%
ważone	222	69,4%
ważone odwrotnie	269	84,1%

dla dopasowania bajtowego. Zatem taki rodzaj funkcji dopasowania przyjęto w dalszej części eksperymentu.

Drugim parametrem, którego rodzaj testowano, było otoczenie. Zaproponowano i rozpatrzono następujące rodzaje otoczenia (podrozdz. 4.1):

1. wszystkie pary:
   - rozmiar potencjalnego otoczenia: $\binom{10}{2} = 45$,
   - rozmiar otoczenia analizowanego w danej iteracji: 45,
2. połowa par:
   - rozmiar potencjalnego otoczenia: 45,
   - rozmiar otoczenia analizowanego w danej iteracji: $\lfloor 45/2 \rfloor = 22$,
3. elementy losowo:
   - rozmiar potencjalnego otoczenia: 45,
   - rozmiar otoczenia analizowanego w danej iteracji: 10,
4. elementy sąsiadująco:
   - rozmiar potencjalnego otoczenia: 10,
   - rozmiar otoczenia analizowanego w danej iteracji: 10.

Na tym etapie wykonano 1 280 testów (4 rodzaje otoczenia dla 10 kluczy i 32 powtórzeń). Liczbę testów, które zakończyły się sukcesem, umieszczono w tab. 6.3. Najlepsze wyniki

Tabela 6.3: Wartości otoczenia dla algorytmu przeszukiwania z tabu (szyfr RC4_10)

Otoczenie	Liczba sukcesów	
wszystkie pary	4	1,3%
połowa par	318	99,4%
elementy losowo	**320**	**100,0%**
elementy sąsiadująco	0	0,0%

(największą liczbę testów, w których odnaleziono poszukiwaną permutację) uzyskano dla otoczenia „elementy losowo". Co więcej, jest to jedyne otoczenie, w którym wszystkie testy zakończyły się sukcesem. Zatem taki rodzaj otoczenia przyjęto w dalszej części eksperymentu.

Kolejnym parametrem, którego wartości testowano, był horyzont. Parametr ten określa liczbę iteracji, przez które dany ruch jest zabroniony. Zaproponowano i rozpatrzono następujące rodzaje horyzontu:

1. $\ln |R(r)| = 4$ – wartość mniejsza niż $\sqrt{|R(r)|}$,
2. $\log_2 |R(r)| = 5$ – wartość mniejsza niż $\sqrt{|R(r)|}$,
3. $\sqrt{|R(r)|} = 7$ – na podstawie [Mic06],
4. $|R(r)|/2 = 22$ – wartość większa niż $\sqrt{|R(r)|}$,
5. $3|S| = 30$ – wartość użyta w problemie komiwojażera [Kno94], który również może być rozpatrywany jako permutacja,
6. losowy z przedziału $\langle \log_2 |R(r)|, 3|S| \rangle = \langle 5, 30 \rangle$,

gdzie:

- $|R(r)|$ – rozmiar otoczenia rozwiązania r,
- $|S|$ – rozmiar permutacji.

Na tym etapie wykonano 1 920 testów (4 rodzaje otoczenia dla 10 kluczy i 32 powtórzeń). Wszystkie testy zakończyły się sukcesem, w związku z tym porównano średnią liczbę permutacji sprawdzonych przed odnalezieniem permutacji poszukiwanej. Rezultaty tych testów zaprezentowano w tab. 6.4. Najlepsze wyniki (najszybszą zbieżność do optimum) uzyskano dla horyzontu $\sqrt{|R(r)|}$. Zatem taką wartość tego parametru przyjęto w dalszej części eksperymentu.

Tabela 6.4: Wartości horyzontu dla algorytmu przeszukiwania z tabu (szyfr RC4_10)

Horyzont	Liczba sukcesów		Liczba sprawdzonych permutacji ($śr.$)
$\ln \|R(r)\|$	**320**	**100,0%**	145 241
$\log_2 \|R(r)\|$	**320**	**100,0%**	159 061
$\sqrt{\|R(r)\|}$	**320**	**100,0%**	**141 613**
$\|R(r)\|/2$	**320**	**100,0%**	167 319
$3\|S\|$	**320**	**100,0%**	142 217
losowy	**320**	**100,0%**	169 402

Na końcu sprawdzono kryterium aspiracji, które pozwala na zaakceptowanie ruchu znajdującego się na liście tabu, o ile prowadzi on do rozwiązania lepszego niż znalezione dotychczas. Zaproponowano i rozpatrzono następujące rodzaje kryterium aspiracji:

1. brak aspiracji – ruchy znajdujące się na liście tabu są bezwzględnie zakazane do wykonania,

2. kryterium aspiracji najlepszego – z ruchu może zostać zdjęty zakaz wykonania, o ile prowadzi do rozwiązania lepszego niż najlepsze znalezione do tej pory.

Na tym etapie wykonano 640 testów (2 rodzaje aspiracji dla 10 kluczy i 32 powtórzeń). Wszystkie wykonane testy zakończyły się sukcesem, w związku z tym porównano średnią liczbę permutacji sprawdzonych przed odnalezieniem permutacji poszukiwanej. Rezultaty tych testów zaprezentowano w tab. 6.5. Najlepsze wyniki (najszybszą zbieżność do optimum) uzyskano dla braku aspiracji. Zatem taką wartość tego parametru przyjęto w dalszej części eksperymentu.

Tabela 6.5: Wartości kryterium aspiracji dla algorytmu przeszukiwania z tabu (szyfr RC4_10)

Aspiracja	Liczba sukcesów		Liczba sprawdzonych permutacji ($śr.$)
brak	**320**	**100,0%**	**141 613**
najlepszego	**320**	**100,0%**	156 677

Wniosek

Na podstawie tego eksperymentu na szyfrze RC4_10 dobrano dla przeszukiwania z tabu następujący zestaw parametrów:

- funkcja dopasowania: dopasowanie bajtowe (wzór (4.4), str. 72),
- otoczenie: elementy losowo,
- horyzont: $\sqrt{|R(r)|}$ (gdzie $|R(r)|$ to rozmiar otoczenia rozwiązania r),
- kryterium aspiracji: brak.

Dla tych wartości parametrów poszukiwaną permutację znaleziono w 320 z 320 testach (100,0%) po sprawdzeniu średnio 141 613 permutacji, co jest równoważne przeszukaniu 3,9% całej przestrzeni rozwiązań. Tych wartości parametrów użyto w kolejnych eksperymentach dotyczących kryptoanalizy szyfru RC4 przy wykorzystaniu algorytmu przeszukiwania z tabu (pkt. 6.1.4, 6.1.6, 6.1.7).

6.1.2 Dobór wartości parametrów dla algorytmu mrówki wierzchołkowej

Celem eksperymentu było sprawdzenie algorytmu mrówki wierzchołkowej w kryptoanalizie szyfru RC4_10 oraz dobór optymalnych parametrów algorytmu. Warunki przeprowadzenia eksperymentu – szczegółowo opisane w dalszej części – były następujące:

- badany algorytm: algorytm mrówki wierzchołkowej (alg. 5.1),
- algorytm szyfrujący poddany kryptoanalizie: RC4_10,
- długość analizowanego strumienia szyfrującego: 10 bajtów,
- 10 kluczy (tab. 6.1),
- warunek zatrzymania: sprawdzenie 3 628 880 permutacji lub $f_{fit} = 100\%$,
- losowa inicjacja początkowej permutacji,
- 4 rodzaje funkcji dopasowania (wzory (4.4), (4.6), (4.8), (4.10), str. 72-73),
- 3 wartości śladu feromonowego maksymalnego,
- 3 wartości śladu feromonowego domyślnego,
- 3 wartości odparowania feromonu,
- 3 rodzaje budowy rozwiązań,
- liczba niezależnych testów dla jednego zestawu parametrów: 32.

Łącznie przeprowadzono 5 120 niezależnych testów $(10 \cdot (4 + 3 + 3 + 3 + 3) \cdot 32)$.

Dobór wartości parametrów algorytmu mrówki wierzchołkowej wykonano według tego samego schematu co w przypadku algorytmu przeszukiwania z tabu (pkt 6.1.1, str. 93). Przyjęto następujące wartości początkowe parametrów:

- funkcja dopasowania: dopasowanie bajtowe (wzór (4.4), str. 72),
- $\tau_{max} = 2|S|$ (gdzie $|S|$ to rozmiar permutacji),
- $\tau_0 = 1$,
- $\rho = 1$,
- budowa rozwiązań: od początku.

Jeden test trwał około 4,5 sekundy.

Pierwsze testy wykonano dla funkcji dopasowania. Zaproponowano i rozpatrzono następujące rodzaje funkcji dopasowania: bajtowe, bajtowe z sąsiadami, ważone, ważone odwrotnie. Na tym etapie wykonano 1 280 testów (4 rodzaje funkcji dopasowania dla 10 kluczy i 32 powtórzeń). Liczbę testów, które zakończyły się sukcesem, tj. znaleziono poszukiwaną permutację przed osiągnięciem maksymalnej liczby sprawdzonych permutacji, umieszczono w tab. 6.6. Najlepsze wyniki (największą liczbę testów, w których odnaleziono poszukiwaną permutację) uzyskano dla funkcji dopasowania ważonego odwrotnie. Zatem taką funkcję dopasowania przyjęto w dalszej części eksperymentu.

Tabela 6.6: Wartości funkcji dopasowania dla algorytmu mrówki wierzchołkowej (szyfr RC4_10)

Dopasowanie	Liczba sukcesów	
bajtowe	183	57,2%
bajtowe z sąsiadami	207	64,7%
ważone	172	53,4%
ważone odwrotnie	**217**	**67,8%**

Drugim parametrem, którego wartości testowano, był parametr τ_{max}, odpowiadający za maksymalną wartość śladu feromonowego, jaki może zostać odłożony na mapie feromonowej w wyniku akumulacji w kolejnych iteracjach. Zaproponowano i rozpatrzono następujące wartości $\tau_{max} = \{2|S| = 20, 4|S| = 40, \infty\}$. Na tym etapie wykonano 960 testów (3 wartości τ_{max} dla 10 kluczy i 32 powtórzeń). Liczbę testów, które zakończyły się sukcesem, umieszczono w tab. 6.7. Najlepsze wyniki (największą liczbę testów, w których odnaleziono daną permutację) uzyskano dla $\tau_{max} = 2|S|$. Zatem taką wartość tego parametru przyjęto w dalszej części eksperymentu.

Tabela 6.7: Wartości parametru τ_{max} dla algorytmu mrówki wierzchołkowej (szyfr RC4_10)

τ_{max}	Liczba sukcesów	
$2\|S\|$	**217**	**67,8%**
$4\|S\|$	55	17,2%
∞ (brak)	1	0,3%

Kolejnym parametrem, którego wartości testowano, był parametr τ_0. Parametr ten odpowiada za inicjalną wartość śladu feromonowego. Zaproponowano i rozpatrzono następujące wartości $\tau_0 = \{1, \tau_{max}/2 = 10, \tau_{max} = 20\}$. Na tym etapie wykonano 960 testów (3 wartości τ_0 dla 10 kluczy i 32 powtórzeń). Liczbę testów, które zakończyły się sukcesem, umieszczono w tab. 6.8. Najlepsze wyniki (największą liczbę testów, w których odnaleziono poszukiwaną permutację) uzyskano dla $\tau_0 = \tau_{max}$. Zatem taką wartość tego parametru przyjęto w dalszej części eksperymentu.

Tabela 6.8: Wartości parametru τ_0 dla algorytmu mrówki wierzchołkowej (szyfr RC4_10)

τ_0	Liczba sukcesów	
1	217	67,8%
$\tau_{max}/2$	220	68,8%
τ_{max}	**231**	**72,2%**

W dalszym toku eksperymentu analizowano wartości parametru ρ, który odpowiada za odparowanie feromonu po każdym przejściu mrówek. Zaproponowano i rozpatrzono następujące wartości $\rho = \{1, |S|/2 = 5, |S| = 10\}$. Na tym etapie wykonano 960 testów (3 wartości ρ dla 10 kluczy i 32 powtórzeń). Liczbę testów, które zakończyły się sukcesem, umieszczono w tab. 6.9. Najlepsze wyniki (największą liczbę testów, w których odnaleziono poszukiwaną permutację) uzyskano dla $\rho = 1$. Zatem taką wartość tego parametru przyjęto w dalszej części eksperymentu.

Tabela 6.9: Wartości parametru ρ dla algorytmu mrówki wierzchołkowej (szyfr RC4_10)

ρ	Liczba sukcesów	
1	**231**	**72,2%**
$\|S\|/2$	146	45,6%
$\|S\|$	206	64,4%

Ostatnim parametrem, którego wartości testowano, był rodzaj budowy rozwiązań. Zaproponowano i rozpatrzono następujące rodzaje budowy rozwiązań: od początku, od końca, losowo. Na tym etapie wykonano 960 testów (3 rodzaje budowy rozwiązań dla 10 kluczy i 32 powtórzeń). Liczbę testów, które zakończyły się sukcesem, umieszczono w tab. 6.10. Najlepsze wyniki (największą liczbę testów, w których odnaleziono poszukiwaną permutację) uzyskano dla budowy rozwiązań losowo. Zatem taką wartość tego parametru przyjęto w dalszej części eksperymentu.

Tabela 6.10: Wartości budowy rozwiązań dla algorytmu mrówki wierzchołkowej (szyfr RC4_10)

Budowa rozwiązań	Liczba sukcesów	
od początku	231	72,2%
od końca	224	70,0%
losowo	**246**	**76,9%**

Wniosek

Najlepsze wyniki algorytmu mrówki wierzchołkowej dla szyfru RC4_10 osiągnięto dla następujących wartości parametrów:

- funkcja dopasowania: dopasowanie ważone odwrotnie (wzór (4.10), str. 73),
- $\tau_{max} = 2|S|$ (gdzie $|S|$ to rozmiar permutacji),
- $\tau_0 = \tau_{max}$,
- ρ = 1,
- budowa rozwiązań: losowo.

Przy takich ustawieniach 76,9% testów zakończyło się sukcesem. Wynik ten jest znacznie niższy niż podczas użycia przeszukiwania z tabu (pkt 6.1.1), gdzie osiągnięto 100% sukcesów. Na podstawie tych badań odrzucono możliwość efektywnego wykorzystania algorytmu mrówki wierzchołkowej w kryptoanalizie szyfru RC4.

6.1.3 Dobór wartości parametrów dla algorytmu mrowiskowego

Celem eksperymentu było sprawdzenie algorytmu mrowiskowego w kryptoanalizie szyfru RC4_10 oraz dobór optymalnych parametrów algorytmu. Warunki przeprowadzenia eksperymentu – szczegółowo opisane w dalszej części – były następujące:

- badany algorytm: algorytm mrowiskowy (alg. 5.2),
- algorytm szyfrujący poddany kryptoanalizie: RC4_10,
- długość analizowanego strumienia szyfrującego: 10 bajtów,
- 10 kluczy (tab. 6.1),
- warunek zatrzymania: sprawdzenie 3 628 880 permutacji lub $f_{fit} = 100\%$,
- losowa inicjacja początkowej permutacji,
- 4 rodzaje funkcji dopasowania (wzory (4.4), (4.6), (4.8), (4.10), str. 72-73),
- 5 wartości τ_0,
- 7 wartości β,
- 5 wartości q_0,
- 5 wartości ρ,

- liczba niezależnych testów dla jednego zestawu parametrów: 32.

Łącznie przeprowadzono 8 320 niezależnych testów ($10 \cdot (5 + 7 + 5 + 5) \cdot 32$).

Dobór wartości parametrów algorytmu mrowiskowego wykonano według tego samego schematu co w przypadku algorytmu przeszukiwania z tabu (pkt 6.1.1, str. 93). Przyjęto następujące wartości początkowe parametrów:

- funkcja dopasowania: dopasowanie bajtowe (wzór (4.4), str. 72),
- τ_0 = 0,1,
- β = 1,
- q_0 = 0,1,
- ρ = 0,1.

Jeden test trwał około 0,6 sekundy.

Pierwsze testy wykonano dla różnej postaci funkcji dopasowania. Zaproponowano i rozpatrzono następujące rodzaje funkcji dopasowania: bajtowe, bajtowe z sąsiadami, ważone, ważone odwrotnie. Na tym etapie wykonano 1 280 testów (4 rodzaje funkcji dopasowania dla 10 kluczy i 32 powtórzeń). Liczbę testów, które zakończyły się sukcesem, tj. znaleziono poszukiwaną permutację przed osiągnięciem maksymalnej liczby iteracji, umieszczono w tab. 6.11. Najlepsze wyniki (największą liczbę testów, w których odnaleziono poszukiwaną permutację) uzyskano dla funkcji dopasowania ważonego. Zatem taką funkcję dopasowania przyjęto w dalszej części eksperymentu.

Tabela 6.11: Wartości funkcji dopasowania dla algorytmu mrowiskowego (szyfr RC4_10)

Dopasowanie	Liczba sukcesów	
bajtowe	305	95,3%
bajtowe z sąsiadami	284	88,8%
ważone	**314**	**98,1%**
ważone odwrotnie	293	91,6%

Drugim parametrem, którego wartości testowano, był parametr τ_0, odpowiadający za inicjalną wartość śladu feromonowego. Zaproponowano i rozpatrzono następujące wartości $\tau_0 = \{0,1; 0,01; 0,001; 0,0001; 0,00001\}$. Na tym etapie wykonano 1 600 testów (5 wartości τ_0 dla 10 kluczy i 32 powtórzeń). Liczbę testów, które zakończyły się sukcesem, umieszczono w tab. 6.12. Najlepsze wyniki (największą liczbę testów, w których odnaleziono poszukiwaną permutację) uzyskano dla $\tau_0 = 0,01$. Zatem taką wartość tego parametru przyjęto w dalszej części eksperymentu.

Tabela 6.12: Wartości parametru τ_0 dla algorytmu mrowiskowego (szyfr RC4_10)

τ_0	Liczba sukcesów	
0,1	314	98,1%
0,01	**316**	**98,8%**
0,001	310	96,9%
0,0001	314	98,1%
0,00001	307	95,9%

Kolejnym parametrem, którego wartości testowano, był parametr β. Parametr ten steruje ważnością śladu feromonowego w wyborze kolejnego kroku przez mrówkę. Zaproponowano i rozpatrzono następujące wartości $\beta = \{0,5; 1; 3; 5; 7; 9; 15\}$. Na tym etapie wykonano 2 240 testów (7 wartości β dla 10 kluczy i 32 powtórzeń). Liczbę testów, które zakończyły się sukcesem, tj. znaleziono poszukiwaną permutację przed osiągnięciem maksymalnej liczby iteracji, umieszczono w tab. 6.13. Dla czterech wartości $\beta = \{5; 7; 9; 15\}$ wszystkie wykonane testy zakończyły się sukcesem. Ponieważ dla kryterium liczby sukcesów nie było możliwe wybranie najlepszej wartości parametru β, wyniki porównano pod względem drugiego kryterium, a mianowicie pod względem średniej liczby rozwiązań sprawdzonych przed odnalezieniem rozwiązania prawidłowego. Te rezultaty również zaprezentowano w tab. 6.13. Najlepsze wyniki (najszybszą zbieżność do optimum) uzyskano dla $\beta = 7$. Zatem taką wartość tego parametru przyjęto w dalszej części eksperymentu.

Tabela 6.13: Wartości parametru β dla algorytmu mrowiskowego (szyfr RC4_10)

β	Liczba sukcesów		Liczba sprawdzonych permutacji ($śr.$)
0,5	293	91,6%	–
1	308	96,3%	–
3	319	99,7%	–
5	**320**	**100,0%**	253 746
7	**320**	**100,0%**	**207 254**
9	**320**	**100,0%**	229 591
15	**320**	**100,0%**	229 757

W dalszym toku eksperymentu analizowano wartości parametru q_0, który odpowiada za ustaleniem zależności pomiędzy eksploracją a eksploatacją. Zaproponowano i rozpatrzono następujące wartości $q_0 = \{0,1; 0,3; 0,5; 0,7; 0,9\}$. Na tym etapie wykonano 1 600 testów (5 wartości q_0 dla 10 kluczy i 32 powtórzeń). Liczbę testów, które zakończyły się sukcesem, umieszczono w tab. 6.14. Dla dwóch wartości $q_0 = \{0,1; 0,3\}$ wszystkie wykonane testy zakończyły się sukcesem. W związku z tym, aby wybrać wartość parametru q_0 dającą najlepsze wyniki, porównano dla tych wartości średnią liczbę rozwiązań sprawdzonych

Tabela 6.14: Wartości parametru q_0 dla algorytmu mrowiskowego (szyfr RC4_10)

q_0	Liczba sukcesów		Liczba sprawdzonych permutacji ($śr.$)
0,1	**320**	**100,0%**	**237 072**
0,3	**320**	**100,0%**	365 588
0,5	319	99,7%	–
0,7	317	99,1%	–
0,9	280	87,5%	–

przed odnalezieniem rozwiązania prawidłowego. Te rezultaty również zaprezentowano w tab. 6.14. Najlepsze wyniki (najszybszą zbieżność do optimum) uzyskano dla $q_0 = 0,1$. Zatem taką wartość tego parametru przyjęto w dalszej części eksperymentu.

Ostatnim parametrem, którego wartości testowano, był parametr ρ, odpowiedzialny za odparowanie feromonu po każdym przejściu mrówek. Zaproponowano i rozpatrzono następujące wartości $\rho = \{0,1; 0,3; 0,5; 0,7; 0,9\}$. Na tym etapie wykonano 1 600 testów (5 wartości ρ dla 10 kluczy i 32 powtórzeń). Wszystkie wykonane testy zakończyły się sukcesem. W związku z tym, aby wybrać wartość parametru ρ dającą najlepsze wyniki, porównano średnią liczbę rozwiązań sprawdzonych przed odnalezieniem rozwiązania prawidłowego. Rezultaty tych testów zaprezentowano w tab. 6.15. Najlepsze wyniki (najszybszą zbieżność do optimum) uzyskano dla $\rho = 0,9$. Zatem taką wartość tego parametru przyjęto w dalszej części eksperymentu.

Tabela 6.15: Wartości parametru ρ dla algorytmu mrowiskowego (szyfr RC4_10)

ρ	Liczba sukcesów		Liczba sprawdzonych permutacji ($śr.$)
0,1	**320**	**100,0%**	254 751
0,3	**320**	**100,0%**	272 035
0,5	**320**	**100,0%**	278 924
0,7	**320**	**100,0%**	270 801
0,9	**320**	**100,0%**	**251 778**

Wniosek

Podsumowując, w wyniku dostrajania parametrów algorytmu mrowiskowego dla szyfru RC4_10 ustalono następujące ich wartości:

- funkcja dopasowania: dopasowanie ważone (wzór (4.8), str. 73),
- τ_0 = 0,01,

- $\beta = 7$,
- $q_0 = 0{,}1$,
- $\rho = 0{,}9$.

Dla tych wartości parametrów poszukiwaną permutację znaleziono w 320 z 320 testów (100,0%) po sprawdzeniu średnio 251 778 permutacji, co jest równoważne przeszukaniu 6,9% całej przestrzeni rozwiązań. Te wartości parametrów użyto w kolejnym eksperymencie dotyczącym kryptoanalizy szyfru RC4 przy wykorzystaniu algorytmu mrowiskowego (pkt 6.1.5).

6.1.4 Algorytm przeszukiwania z tabu w kryptoanalizie szyfru RC4_16

Celem eksperymentu było sprawdzenie możliwości kryptoanalizy proponowaną w rozprawie metodą większej niż w poprzednich eksperymentach wersji szyfru RC4. Do badań wybrano wersję szyfru, w której założono, że permutacja przechowywana w tablicy S składa się z szesnastu liczb z zakresu od 0 do 15. Oznaczono tę wersję jako RC4_16. Zasada działania algorytmu jest taka sama jak w alg. 2.1, z jedyną różnicą taką, że wszystkie operacje wykonywane są $\bmod 16$. Liczba możliwych kluczy dla tej wielkości szyfru RC4 to 2^{128}, a liczba możliwych początkowych stanów wewnętrznych to $16! = 20\,922\,789\,888\,000 \approx 2^{44}$.

Warunki przeprowadzenia eksperymentu były następujące:

- badany algorytm: przeszukiwanie z tabu (alg. 4.1),
- algorytm szyfrujący poddany kryptoanalizie: RC4_16,
- długość analizowanego strumienia szyfrującego: 16 bajtów,
- 5 kluczy (tab. 6.16),
- warunek zatrzymania: sprawdzenie 1 600 000 000 permutacji lub $f_{fit} = 100\%$,
- losowa inicjacja początkowej permutacji,
- parametry dobrane zgodnie z wynikami eksperymentu z pkt. 6.1.1 (tab. 6.17),
- liczba niezależnych testów dla jednego zestawu parametrów: 32.

Łącznie przeprowadzono 160 testów ($5 \cdot 1 \cdot 32$).

Dla tego rodzaju otoczenia („elementy losowo”) oraz permutacji o wielkości 16 bajtów rozmiar potencjalnego otoczenia to $\binom{16}{2} = 120$, a rozmiar otoczenia analizowanego w danej iteracji: 16.

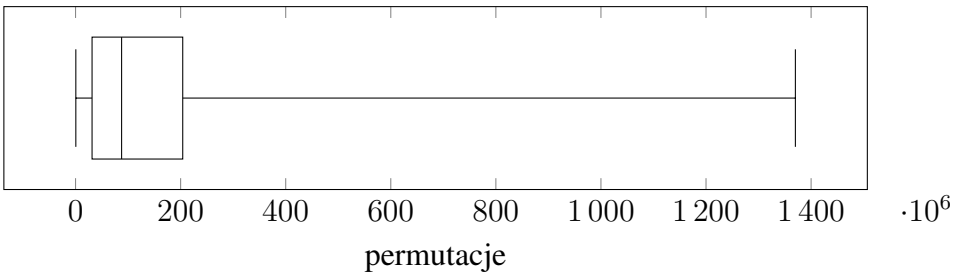
Tabela 6.16: Klucze użyte w kryptoanalizie szyfru RC4_16

Klucz
0x0102030405
0x010203040506070809a
0x010203040506070809a0b0c0d0e0f00
0x0900030202070a
0x0b040207060c0b07040900030202070a

Tabela 6.17: Wartości parametrów użyte w kryptoanalizie szyfru RC4_16

Dopasowanie	Horyzont	Otoczenie	Kryterium aspiracji
bajtowe	$\sqrt{\lvert R(r)\rvert} = 11$	elementy losowo	brak

Z przeprowadzonych 160 testów 157 (98,1%) zakończyło się odnalezieniem prawidłowego stanu wewnętrznego w zadanej liczbie sprawdzonych permutacji. Na rys. 6.1 przedstawiono w formie wykresu pudełkowego liczbę permutacji sprawdzonych przed znalezieniem permutacji poszukiwanej. Na wykresie uwzględniono tylko te testy, które zakończyły się sukcesem, tj. odnaleziono poszukiwaną permutację. Średnio potrzebne było sprawdzenie 185 403 978 permutacji przed odnalezieniem permutacji poszukiwanej. Stanowi to 0,0009% całej przestrzeni rozwiązań. Jeden test trwał około 2 minuty i 16 sekund.



Rysunek 6.1: Liczba permutacji sprawdzonych przed znalezieniem prawidłowej permutacji szyfru RC4_16

Wniosek

Proponowana metoda kryptoanalizy przy użyciu przeszukiwania z tabu jest skuteczna dla szyfru RC4 o rozmiarze 16.

6.1.5 Algorytm mrowiskowy w kryptoanalizie szyfru RC4_16

Celem eksperymentu było sprawdzenie możliwości kryptoanalizy proponowanym w rozprawie algorytmem mrowiskowym większej niż poprzednio (pkt 6.1.3) wersji szyfru RC4. Warunki przeprowadzenia eksperymentu były następujące:

- badany algorytm: algorytm mrowiskowy (alg. 5.2),
- algorytm szyfrujący poddany kryptoanalizie: RC4_16,
- długość analizowanego strumienia szyfrującego: 16 bajtów,
- 5 kluczy (tab. 6.16),
- warunek zatrzymania: sprawdzenie 1 600 000 000 permutacji lub $f_{fit} = 100\%$,
- losowa inicjacja początkowej permutacji,
- wartości parametrów zgodne z wynikami eksperymentu z pkt. 6.1.3:
  - funkcja dopasowania: dopasowanie ważone (wzór (4.8), str. 73),
  - τ_0 = 0,01,
  - β = 7,
  - q_0 = 0,1,
  - ρ = 0,9,
- liczba niezależnych testów dla jednego zestawu parametrów: 32.

Łącznie przeprowadzono 160 testów ($5 \cdot 1 \cdot 32$).

Z przeprowadzonych 160 testów 45 (28,1%) zakończyło się odnalezieniem prawidłowego stanu wewnętrznego w zadanej liczbie iteracji. W przypadku testów, które zakończyły się sukcesem, tj. odnalezieniem poszukiwanej permutacji, potrzebne było sprawdzenie średnio 802 324 675 permutacji przed odnalezieniem permutacji poszukiwanej. Stanowi to 0,0038% całej przestrzeni przeszukiwań. Jeden test trwał około 17 minut i 37 sekund.

Wniosek

Przy użyciu algorytmu mrowiskowego uzyskano w zadanej liczbie sprawdzonych permutacji 3 razy mniej sukcesów niż przy użyciu przeszukiwania z tabu (6.1.4). Co więcej, testy, które zakończyły się sukcesem, potrzebowały sprawdzenia 4,5 razy większej liczby permutacji w przypadku algorytmu mrowiskowego w porównaniu do przeszukiwania z tabu. Zatem proponowana metoda kryptoanalizy przy użyciu algorytmu mrowiskowego nie wydaje się być skuteczna dla szyfru RC4 o rozmiarze 16.

6.1.6 Algorytm przeszukiwania z tabu w kryptoanalizie szyfru RC4

W eksperymencie przeprowadzono proponowany atak kryptoanalityczny na najbardziej znany szyfr strumieniowy RC4. Warunki przeprowadzenia eksperymentu były następujące:

- badany algorytm: przeszukiwanie z tabu (alg. 4.1),
- algorytm szyfrujący poddany kryptoanalizie: RC4,
- długość analizowanego strumienia szyfrującego: 256 bajtów,
- 10 kluczy (tab. 6.18) wybranych z oficjalnych wektorów testowych [SJ11],
- warunek zatrzymania: 19 531 iteracji,
- losowa inicjacja początkowej permutacji,
- parametry dobrane zgodnie z wynikami eksperymentu z pkt. 6.1.1 (tab. 6.17),
- liczba niezależnych testów dla jednego zestawu parametrów: 32.

Łącznie przeprowadzono 320 testów ($10 \cdot 1 \cdot 32$).

Wygenerowane losowo 5 000 000 permutacji oceniono za pomocą tej samej funkcji dopasowania. Jest to w przybliżeniu równe liczbie permutacji ocenionych w trakcie jednego testu przeszukiwania z tabu (19 531 iteracji analizujących po 256 rozwiązań to 4 999 936 rozpatrzonych rozwiązań). Liczbę iteracji algorytmu przeszukiwania z tabu dobrano tak, by możliwe było porównanie z [Fer13, FO14] (pkt 6.1.7).

Dla tego rodzaju otoczenia („elementy losowo") oraz permutacji o wielkości 256 bajtów rozmiar potencjalnego otoczenia to $\binom{256}{2} = 32\,640$, a rozmiar otoczenia analizowanego

Tabela 6.18: Klucze użyte w kryptoanalizie szyfru RC4 (wybrane z oficjalnych wektorów testowych [SJ11])

ID	Klucz
1-0	0x0102030405
1-1	0x01020304050607
1-2	0x0102030405060708090a
1-3	0x0102030405060708090a0b0c0d0e0f10
1-4	0x0102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f20
1-5	0x833222772a
1-6	0x1910833222772a
1-7	0x8b37641910833222772a
1-8	0xebb46227c6cc8b37641910833222772a
1-9	0xc109163908ebe51debb46227c6cc8b37641910833222772a

w danej iteracji: 256. Przy tym rozmiarze permutacji horyzont $\sqrt{|R(r)|}$ = 181 (gdzie $|R(r)|$ to rozmiar otoczenia rozwiązania r).

Wyniki eksperymentu zostały zebrane w tab. 6.19. Każdy przedstawiony w tabeli wynik jest parą składającą się z wartości funkcji dopasowania dla najlepszego stanu wewnętrznego (u góry) i numeru iteracji, w której ten najlepszy stan wewnętrzny został znaleziony (na dole). W wierszach znajdują się wyniki osobno dla każdego klucza (zob. tab. 6.18), a także sumaryczne wyniki dla wszystkich kluczy (na samym dole). W ostatniej kolumnie zaprezentowane są wyniki dla losowo generowanych stanów wewnętrznych, z daną funkcją dopasowania. Dla każdego zestawu klucz-horyzont umieszczono w tabeli najgorszy, średni oraz najlepszy wynik, a także odchylenie standardowe otrzymane z 32 wykonań algorytmu. W przypadku rozwiązań generowanych losowo jest to najgorszy, średni i najlepszy wynik oraz odchylenie standardowe otrzymane z 5 000 000 prób.

Rezultaty dla przedstawionego zestawu parametrów to funkcja dopasowania o wartości średnio 26,9% przy dopasowaniu bajtowym. Rezultaty są porównywalne dla różnych kluczy, więc nie zależą od szczególnych właściwości klucza. Otrzymane rezultaty są również dość stabilne i dobrze skupione, ponieważ odchylenie standardowe jest niewielkie ($\leqslant \ 2,0\%$). Strumień szyfrujący wygenerowany przez losowe permutacje osiągnął wartość funkcji dopasowania maksymalnie 3,9% (średnio 0,4%) dla dopasowania bajtowego. Są to wartości dużo mniejsze niż uzyskane za pomocą przeszukiwania z tabu. Jeden test trwał około 37 sekund.

Aproksymację przebiegu funkcji dopasowania sprawdzono za pomocą analizy regresji. Sprawdzono cztery typy regresji: liniową, logarytmiczną, wykładniczą i potęgową. Regresja logarytmiczna daje najmniejszą wartość błędu R^2, dlatego w dalszej analizie wyniki przedstawiane będą za jej pomocą.

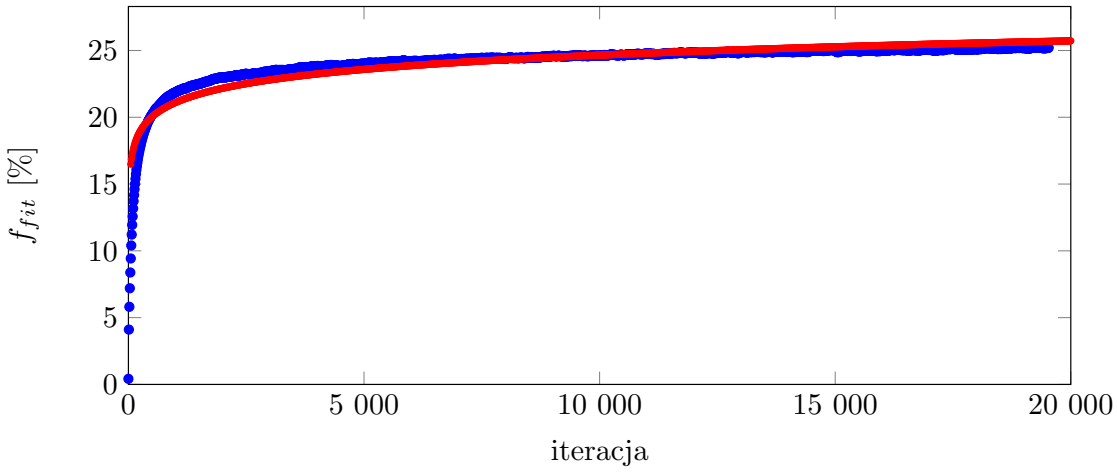
Uśredniony przebieg funkcji dopasowania w kolejnych iteracjach przedstawiono na rys. 6.2 kolorem niebieskim. Na wykresie zaznaczono również kolorem czerwonym linię regresji logarytmicznej odpowiadającej przedstawionemu przebiegowi funkcji dopasowania. Równanie regresji logarytmicznej dla tego wykresu można opisać wzorem:

$$f(x) = 0,0154102458 \ln(x) + 0,1045792553 \qquad (R^2 = 0,829), \tag{6.1}$$

gdzie R^2 to współczynnik determinacji. Korzystając z ekstrapolacji można na podstawie powyższego wzoru obliczyć, że pełne pokrycie strumienia szyfrującego uda się osiągnąć po około 2^{84} iteracji. W każdej iteracji sprawdzanych jest 2^8 rozwiązań. Zatem w celu odnalezienia stanu wewnętrznego trzeba sprawdzić około 2^{92} możliwości.

Tabela 6.19: Wyniki uzyskane dla szyfru RC4; w przypadku algorytmu przeszukiwania z tabu dla każdego klucza podano wartość funkcji dopasowania oraz (poniżej) numer iteracji, w której tę wartość uzyskano. Dla permutacji wygenerowanych losowo dla każdego klucza podano uzyskaną wartość funkcji dopasowania w podanej liczbie prób. Użyte oznaczenia: min – wartość minimalna, $śr.$ – średnia arytmetyczna, max – wartość maksymalna, s – odchylenie standardowe

Dopasowanie	bajtowe							
Algorytm	przeszukiwanie z tabu				losowo			
ID klucza	min	$śr.$	max	s	min	$śr.$	max	s
1-0	23,4%	26,9%	29,3%	1,4%	0,0%	0,4%	3,9%	0,4%
	3 751	12 914	19 100	4 428	–	–	–	–
1-1	23,8%	26,9%	29,7%	1,6%	0,0%	0,4%	3,9%	0,4%
	3 927	13 839	19 501	4 608	–	–	–	–
1-2	24,6%	26,6%	28,9%	1,1%	0,0%	0,4%	3,9%	0,4%
	6 548	13 256	19 270	3 927	–	–	–	–
1-3	24,2%	27,4%	34,0%	2,0%	0,0%	0,4%	3,5%	0,4%
	7 091	13 318	19 254	3 771	–	–	–	–
1-4	23,8%	27,1%	30,1%	1,6%	0,0%	0,4%	3,9%	0,4%
	5 574	13 103	19 450	4 218	–	–	–	–
1-5	24,2%	26,8%	29,7%	1,4%	0,0%	0,4%	3,9%	0,4%
	6 688	12 826	18 644	3 630	–	–	–	–
1-6	21,5%	26,7%	30,9%	2,0%	0,0%	0,4%	3,5%	0,4%
	8 036	14 486	19 340	3 579	–	–	–	–
1-7	24,2%	27,3%	30,1%	1,7%	0,0%	0,4%	3,5%	0,4%
	4 949	13 118	19 319	3 901	–	–	–	–
1-8	23,0%	26,7%	30,1%	1,5%	0,0%	0,4%	3,5%	0,4%
	7 132	13 348	19 389	3 528	–	–	–	–
1-9	23,8%	27,1%	30,5%	1,6%	0,0%	0,4%	3,9%	0,4%
	5 175	14 788	19 262	3 485	–	–	–	–
razem	21,5%	26,9%	34,0%	1,6%	0,0%	0,4%	3,9%	0,4%
	3 751	13 500	19 501	3 920	–	–	–	–



Rysunek 6.2: Wykres średniej wartości funkcji dopasowania w kolejnych iteracjach przeszukiwania z tabu (szyfr RC4) – niebieskie punkty; wykres regresji logarytmicznej zaznaczono kolorem czerwonym

Wniosek

Przeszukiwanie z tabu daje gorsze wyniki niż najlepszy znany atak na szyfr RC4 (pkt 2.4.1). Z tego powodu poddano kryptoanalizie inne szyfry strumieniowe o podobnej budowie (podrozdz. 6.2 i 6.3). Są to szyfry VMPC oraz RC4+, których słabe strony nie zostały w wystarczający sposób zbadane.

6.1.7 Porównanie wyników z innymi metaheurystykami

W niniejszym badaniu porównano działanie różnych metaheurystyk. Poza proponowanym w rozprawie atakiem z użyciem przeszukiwania z tabu są to: algorytm genetyczny (GA), symulowane wyżarzanie (SA) oraz optymalizacja stadna cząsteczek (PSO), dla których wyniki opracowano na podstawie [Fer13, FO14]. Eksperyment przeprowadzono na algorytmie RC4, który jest najbardziej znanym szyfrem strumieniowym. Wyniki dla algorytmu przeszukiwania z tabu opracowano na podstawie pkt. 6.1.6.

Wyniki zostały zebrane w tab. 6.20. W kolejnych wierszach tabeli przedstawione są kolejne algorytmy metaheurystyczne: GA w dwóch wariantach (nieadaptacyjnym i adaptacyjnym), PSO, SA oraz TS. W trzech ostatnich wierszach umieszczono najlepsze, średnie oraz najgorsze wyniki uzyskane podczas losowego generowania permutacji. Ponieważ każdy z rozpatrywanych w niniejszym punkcie algorytmów w danej iteracji (pokoleniu) sprawdza różną liczbę możliwych rozwiązań, porównano, ile całościowo rozwiązań zostało sprawdzonych w trakcie wykonania danego algorytmu. W pierwszej kolumnie znajduje się nazwa algorytmu. W drugiej oraz trzeciej kolumnie znajdują się odpowiednio: liczba iteracji danego algorytmu oraz liczba rozwiązań rozpatrywanych w jednej iteracji (wielkość iteracji). Te liczby mnożone przez siebie dają sumaryczną liczbę

Tabela 6.20: Porównanie kryptoanalizy szyfru RC4 z użyciem różnych metaheurystyk (⋆ wyniki opracowane na podstawie [Fer13, FO14])

Algorytm	Liczba iteracji		Wielkość iteracji	Rozpatrzonych permutacji	f_{fit} $(śr.)$
⋆ GA nieadaptacyjne	1 000 000	·	5	$\approx 2^{22}$	18,1%
⋆ GA adaptacyjne	1 000 000	·	5	$\approx 2^{22}$	20,7%
⋆ PSO	1 000 000	·	10	$\approx 2^{23}$	0,7%
⋆ SA	10 000 000	·	1	$\approx 2^{23}$	3,0%
TS	19 531	·	256	$\approx 2^{22}$	**26,9%**
losowo (min)			5 000 000	$\approx 2^{22}$	0,0%
losowo $(śr.)$			5 000 000	$\approx 2^{22}$	0,4%
losowo (max)			5 000 000	$\approx 2^{22}$	3,9%

sprawdzonych permutacji, która w postaci potęgi dwójki została umieszczona w kolumnie czwartej (wszystkie liczby są rzędu 10^7). W ostatniej kolumnie znajduje się osiągnięta wartość funkcji dopasowania (dopasowanie bajtowe, wzór (4.4), str. 72). Rezultaty dla wszystkich algorytmów są wartościami średnimi.

Rezultaty dla GA, PSO oraz SA opracowano na podstawie [Fer13, FO14]. Autorzy wymienionych prac również poddali kryptoanalizie szyfr RC4 (podrozdz. 3.3), a funkcją dopasowania była procentowa zgodność bajtów w 256-bajtowym strumieniu szyfrującym. Dla każdego algorytmu zostało wykonanych 25 testów. Rezultaty dla TS opracowano na podstawie tab. 6.19.

Jak widać zarówno symulowane wyżarzanie, jak i optymalizacja stadna cząsteczek nie sprawdziły się. Nie są znane przyczyny takiego stanu rzeczy, ponieważ algorytmy te wypadły gorzej niż przeszukiwanie losowe (dla tej samej wielkości próbki przeszukiwanie losowe osiągnęło maksymalną wartość funkcji dopasowania na poziomie 3,9%). Być może przyczyna tkwi w niepoprawnym doborze parametrów lub niedopasowaniu reprezentacji rozwiązania do zadanego problemu.

Autorzy [FO14] podają, że na podstawie wykresu wyników i przy użyciu regresji logarytmicznej znaleźli równanie:

$$f(x) = 0,0131306222 \ln(x) - 0,1065234375. \tag{6.2}$$

Korzystając z tego równania można policzyć, ile generacji średnio będzie potrzebnych do znalezienia prawidłowego rozwiązania. Będzie to około 2^{122} generacji, każda składająca się z 5 ($\approx 2^2$) osobników, czyli razem przeanalizowanych zostanie około 2^{124} możliwych stanów wewnętrznych szyfru RC4.

Na podstawie wyników eksperymentu z pkt. 6.1.6 można wnioskować, że przeszukiwanie z tabu pozwala na znalezienie prawidłowego stanu wewnętrznego

w okolicach 2^{84} iteracji, z czego w każdej iteracji sprawdzanych jest 2^8 rozwiązań. Łącznie daje to sprawdzenie 2^{92} permutacji w czasie działania algorytmu. To znacznie mniej niż w przypadku algorytmu genetycznego przedstawionego przez Ferrimana, gdzie potrzebne było sprawdzenie 2^{124} stanów wewnętrznych.

Wniosek

Przeszukiwanie z tabu sprawdza się w kryptoanalizie szyfru strumieniowego na bazie permutacji lepiej niż zwykłe przeszukiwanie losowe, a także lepiej niż inne metaheurystyki (algorytm genetyczny, symulowane wyżarzanie, optymalizacja stadna cząsteczek).

6.2 Kryptoanaliza szyfru VMPC

W poniższych eksperymentach zbadano możliwość użycia zaproponowanych algorytmów w kryptoanalizie polskiego szyfru strumieniowego VMPC.

6.2.1 Dobór wartości parametrów dla algorytmu przeszukiwania z tabu

Celem eksperymentu było sprawdzenie wpływu poszczególnych parametrów algorytmu na jakość otrzymywanych rozwiązań. Do badań wybrano mniejszą wersję szyfru, w której założono, że permutacja przechowywana w tablicy S składa się z dziesięciu liczb z zakresu od 0 do 9. Wersję tę oznaczono jako VMPC_10. Zasada działania algorytmu jest taka sama jak w alg. 2.2, z tą różnicą, że wszystkie operacje wykonywane są $\mathrm{mod}\ 10$.

Warunki przeprowadzenia eksperymentu oraz jego scenariusz są takie same jak w przypadku użycia przeszukiwania z tabu do kryptoanalizy szyfru RC4_10 (pkt 6.1.1, str. 93). Jedyną różnicą jest to, że szyfrem poddanym kryptoanalizie w obecnym eksperymencie jest szyfr VMPC_10. W związku z tym założenia nie będą tutaj szczegółowo prezentowane. Przedstawione zostaną jedynie wyniki, które zaprezentowano w tab. 6.22-6.25. Jeden test trwał około 0,8 sekundy.

Wniosek

Na podstawie tego eksperymentu na szyfrze VMPC_10 dobrano dla przeszukiwania z tabu następujący zestaw parametrów:

- funkcja dopasowania: dopasowanie bajtowe (wzór (4.4), str. 72),
- otoczenie: połowa par,

Tabela 6.21: Klucze użyte w kryptoanalizie szyfru VMPC_10

Klucz	IV
0x0007050003010609	0x0502070002030309
0x0000000000000000	0x0502070002030309
0x0000000000000000	0x0505050505050505
0x0000000000000000	0x0000000000000000
0x0008000000000000	0x0000000000000800
0x0909090909090909	0x0000000000000000
0x0001020304050607	0x0000000000000000
0x0003060306030609	0x0000000000000000
0x0000000000000000	0x0000000600000000
0x0000000000000000	0x0003060306030609

Tabela 6.22: Wartości funkcji dopasowania dla algorytmu przeszukiwania z tabu (szyfr VMPC_10)

Dopasowanie	Liczba sukcesów	
bajtowe	**302**	**94,4%**
bajtowe z sąsiadami	287	89,7%
ważone	219	68,4%
ważone odwrotnie	206	64,8%

Tabela 6.23: Wartości otoczenia dla algorytmu przeszukiwania z tabu (szyfr VMPC_10)

Otoczenie	Liczba sukcesów	
wszystkie pary	0	0,0%
połowa par	**302**	**94,4%**
elementy losowo	299	93,4%
elementy sąsiadująco	0	0,0%

Tabela 6.24: Wartości horyzontu dla algorytmu przeszukiwania z tabu (szyfr VMPC_10)

Horyzont	Liczba sukcesów	
$\ln \lvert R(r)\rvert$	302	94,4%
$\log_2 \lvert R(r)\rvert$	295	92,2%
$\sqrt{\lvert R(r)\rvert}$	302	94,4%
$\lvert R(r)\rvert/2$	292	91,3%
$3\lvert S\rvert$	**305**	**95,3%**
losowy	297	92,8%

Tabela 6.25: Wartości kryterium aspiracji dla algorytmu przeszukiwania z tabu (szyfr VMPC_10)

Aspiracja	Liczba sukcesów	
brak	**305**	**95,3%**
najlepszego	299	92,2%

- horyzont: $3|S|$ (gdzie $|S|$ to rozmiar permutacji),
- kryterium aspiracji: brak.

Dla tych wartości parametrów poszukiwaną permutację znaleziono w 305 próbach z 320 (95,3%) po sprawdzeniu średnio 858 708 permutacji, co jest równoważne przeszukaniu 23,7% całej przestrzeni rozwiązań. Te wartości parametrów użyto w kolejnych eksperymentach dotyczących kryptoanalizy szyfru VMPC przy wykorzystaniu algorytmu przeszukiwania z tabu (pkt. 6.2.4, 6.2.6).

6.2.2 Dobór wartości parametrów dla algorytmu mrówki wierzchołkowej

Celem eksperymentu było sprawdzenie algorytmu mrówki wierzchołkowej w kryptoanalizie szyfru VMPC_10 oraz dobór optymalnych parametrów algorytmu. Warunki przeprowadzenia eksperymentu oraz jego scenariusz są takie same jak w przypadku użycia algorytmu mrówki wierzchołkowej do kryptoanalizy szyfru RC4_10 (pkt 6.1.2, str. 98). Jedyną różnicą jest to, że szyfrem poddanym kryptoanalizie w aktualnym eksperymencie jest szyfr VMPC_10. W związku z tym założenia nie będą tutaj szczegółowo prezentowane. Przedstawione zostaną jedynie wyniki, które zaprezentowano w tab. 6.26-6.30. Wartości kluczy użytych w badaniach umieszczono w tab. 6.21. Jeden test trwał około 5,3 sekund.

Tabela 6.26: Wartości funkcji dopasowania dla algorytmu mrówki wierzchołkowej (szyfr VMPC_10)

Dopasowanie	Liczba sukcesów	
bajtowe	189	59,1%
bajtowe z sąsiadami	201	62,8%
ważone	**203**	**63,4%**
ważone odwrotnie	198	61,9%

Tabela 6.27: Wartości parametru τ_{max} dla algorytmu mrówki wierzchołkowej (szyfr VMPC_10)

τ_{max}	Liczba sukcesów	
$2\|S\|$	**203**	**63,4%**
$4\|S\|$	44	13,8%
∞ (brak)	0	0,0%

Tabela 6.28: Wartości parametru τ_0 dla algorytmu mrówki wierzchołkowej (szyfr VMPC_10)

τ_0	Liczba sukcesów	
1	203	63,4%
$\tau_{max}/2$	213	66,6%
τ_{max}	**218**	**68,1%**

Tabela 6.29: Wartości parametru ρ dla algorytmu mrówki wierzchołkowej (szyfr VMPC_10)

ρ	Liczba sukcesów		Liczba sprawdzonych permutacji ($śr.$)
1	**218**	**68,1%**	**1 437 887**
$\|S\|/2$	**218**	**68,1%**	1 626 975
$\|S\|$	186	64,4%	–

Tabela 6.30: Wartości budowy rozwiązań dla algorytmu mrówki wierzchołkowej (szyfr VMPC_10)

Budowa rozwiązań	Liczba sukcesów	
od początku	**218**	**68,1%**
od końca	184	57,5%
losowo	193	60,3%

Wniosek

Najlepsze wyniki algorytmu mrówki wierzchołkowej dla szyfru VMPC_10 osiągnięto dla następujących wartości parametrów:

- funkcja dopasowania: dopasowanie ważone odwrotnie (wzór (4.10), str. 73),
- $\tau_{max} = 2|S|$ (gdzie $|S|$ to rozmiar permutacji),
- $\tau_0 = \tau_{max}$,
- ρ = 1,
- budowa rozwiązań: od początku.

Przy takich ustawieniach 68,1% testów zakończyło się sukcesem. Wynik ten jest znacznie niższy niż podczas użycia przeszukiwania z tabu (pkt 6.2.1), gdzie osiągnięto 95,3% sukcesów. Na podstawie tych badań odrzucono możliwość efektywnego wykorzystania algorytmu mrówki wierzchołkowej w kryptoanalizie szyfru VMPC.

6.2.3 Dobór wartości parametrów dla algorytmu mrowiskowego

Celem eksperymentu było sprawdzenie algorytmu mrowiskowego w kryptoanalizie szyfru VMPC_10 oraz dobór optymalnych parametrów algorytmu. Warunki

przeprowadzenia eksperymentu oraz jego scenariusz są analogiczne jak w przypadku użycia algorytmu mrowiskowego do kryptoanalizy szyfru RC4_10 (pkt 6.1.3, str. 101). Jedyną różnicą jest to, że szyfrem poddanym kryptoanalizie w obecnym eksperymencie jest szyfr VMPC_10. W związku z tym założenia nie będą tutaj szczegółowo prezentowane. Przedstawione zostaną jedynie wyniki, które zaprezentowano w tab. 6.31-6.35. Wartości kluczy użytych w badaniach umieszczono w tab. 6.21. Jeden test trwał około 1,3 sekundy.

Tabela 6.31: Wartości funkcji dopasowania dla algorytmu mrowiskowego (szyfr VMPC_10)

Dopasowanie	Liczba sukcesów	
bajtowe	240	75,0%
bajtowe z sąsiadami	219	68,4%
ważone	**254**	**79,8%**
ważone odwrotnie	214	66,9%

Tabela 6.32: Wartości parametru τ_0 dla algorytmu mrowiskowego (szyfr VMPC_10)

τ_0	Liczba sukcesów	
0,1	**212**	**66,3%**
0,01	202	63,1%
0,001	197	61,6%
0,0001	199	62,2%
0,00001	196	61,3%

Tabela 6.33: Wartości parametru β dla algorytmu mrowiskowego (szyfr VMPC_10)

β	Liczba sukcesów	
0,5	241	75,3%
1	272	85,0%
3	292	91,3%
5	304	95,0%
7	**306**	**95,6%**
9	299	93,4%
15	299	93,4%

Wniosek

Podsumowując, w wyniku dostrajania parametrów algorytmu mrowiskowego dla szyfru VMPC_10 ustalono następujące ich wartości:

- funkcja dopasowania: dopasowanie ważone (wzór (4.8), str. 73),

Tabela 6.34: Wartości parametru q_0 dla algorytmu mrowiskowego (szyfr VMPC_10)

q_0	Liczba sukcesów	
0,1	**304**	**95,0%**
0,3	300	93,8%
0,5	284	88,8%
0,7	272	85,0%
0,9	207	64,7%

Tabela 6.35: Wartości parametru ρ dla algorytmu mrowiskowego (szyfr VMPC_10)

ρ	Liczba sukcesów	
0,1	301	94,1%
0,3	296	92,5%
0,5	304	95,0%
0,7	**306**	**95,6%**
0,9	303	94,7%

- $\tau_0 = 0{,}1$,
- $\beta = 7$,
- $q_0 = 0{,}1$,
- $\rho = 0{,}7$.

Dla tych wartości parametrów poszukiwaną permutację znaleziono w 306 z 320 prób (95,6%). W przypadku testów, które zakończyły się sukcesem, potrzebne było sprawdzenie średnio 1 028 672 permutacji przed odnalezieniem tej prawidłowej. Stanowi to 28,3% całej przestrzeni poszukiwań. Tych wartości parametrów użyto w kolejnych eksperymentach dotyczących kryptoanalizy szyfru VMPC przy wykorzystaniu algorytmu mrowiskowego (pkt 6.2.5).

6.2.4 Algorytm przeszukiwania z tabu w kryptoanalizie szyfru VMPC_16

Celem eksperymentu było sprawdzenie możliwości kryptoanalizy proponowaną w rozprawie metodą większej niż w poprzednich eksperymentach wersji szyfru VMPC. Do badań wybrano wersję szyfru, w której założono, że permutacja przechowywana w tablicy S składa się z szesnastu liczb z zakresu od 0 do 15. Oznaczono tę wersję jako VMPC_16. Zasada działania szyfru jest taka sama jak w alg. 2.2, z jedyną różnicą taką, że wszystkie operacje wykonywane są $\mathrm{mod}\ 16$. Liczba możliwych kluczy dla tej

wielkości szyfru VMPC to 2^{128}, a liczba możliwych początkowych stanów wewnętrznych to $16! = 20\,922\,789\,888\,000 \approx 2^{44}$.

Warunki przeprowadzenia eksperymentu były następujące:

- badany algorytm: przeszukiwanie z tabu (alg. 4.1),
- algorytm szyfrujący poddany kryptoanalizie: VMPC_16,
- długość analizowanego strumienia szyfrującego: 16 bajtów,
- 5 kluczy (tab. 6.36),
- warunek zatrzymania: sprawdzenie 1 600 000 000 permutacji lub $f_{fit} = 100\%$,
- losowa inicjacja początkowej permutacji,
- parametry dobrane zgodnie z wynikami eksperymentu z pkt. 6.2.1 (tab. 6.37),
- liczba niezależnych testów dla jednego zestawu parametrów: 32.

Łącznie przeprowadzono 160 testów ($5 \cdot 1 \cdot 32$).

Tabela 6.36: Klucze użyte w kryptoanalizie szyfru VMPC_16

Klucz	IV
0x0601010A070708090B060C01070D0607	0x0B0C0F000E0703050708020F0D020105
0x0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A	0x0505050505050505050505050505050505
0x00000000000000000000000000000000	0x080F060D040B020900070F0205080B0E
0x01000000000000000000000000000000	0x00000000000000000000000000000000
0x00000000000000000000000000000000	0x00000000000000000000000000010000

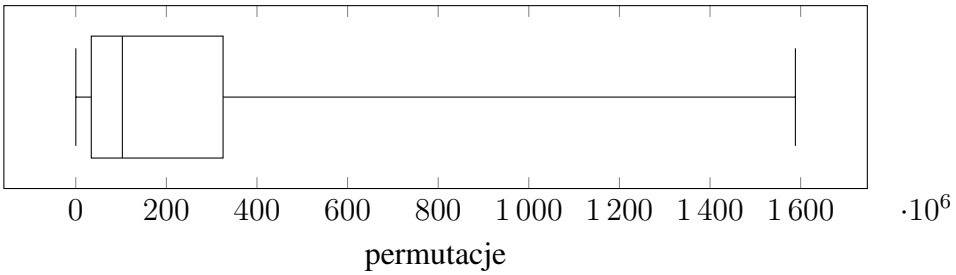
Tabela 6.37: Wartości parametrów użyte w kryptoanalizie szyfru VMPC_16

Dopasowanie	Horyzont	Otoczenie	Kryterium aspiracji
bajtowe	$3\|S\| = 48$	połowa par	brak

Dla tego rodzaju otoczenia („połowa par”) oraz permutacji o wielkości 16 bajtów rozmiar potencjalnego otoczenia to 120 (2 elementy z 16 można wybrać na $\binom{16}{2} = 120$ sposobów), a rozmiar otoczenia analizowanego w danej iteracji: 60 (analizowana jest połowa z możliwych zamian).

Z przeprowadzonych 160 testów 129 (80,6%) zakończyło się odnalezieniem prawidłowej permutacji w zadanej liczbie sprawdzonych permutacji. Na rys. 6.3 przedstawiono w formie wykresu pudełkowego liczbę permutacji sprawdzonych przed znalezieniem permutacji poszukiwanej. Na wykresie uwzględniono tylko te testy, które zakończyły się sukcesem,

tj. odnaleziono poszukiwaną permutację. Średnio potrzebne było sprawdzenie 278 033 351 permutacji przed odnalezieniem permutacji poszukiwanej. Stanowi to 0,0013% całej przestrzeni rozwiązań. Jeden test trwał około 22 minut.



Rysunek 6.3: Liczba permutacji potrzebnych do znalezienia prawidłowej permutacji szyfru VMPC_16

Wniosek

Proponowana metoda kryptoanalizy przy użyciu przeszukiwania z tabu jest skuteczna dla szyfru VMPC o rozmiarze 16.

6.2.5 Algorytm mrowiskowy w kryptoanalizie szyfru VMPC_16

Celem eksperymentu było sprawdzenie możliwości kryptoanalizy proponowanym w rozprawie algorytmem mrowiskowym większej niż poprzednio (pkt 6.2.3) wersji szyfru VMPC. Warunki przeprowadzenia eksperymentu były następujące:

- badany algorytm: algorytm mrowiskowy (alg. 5.2),
- algorytm szyfrujący poddany kryptoanalizie: VMPC_16,
- długość analizowanego strumienia szyfrującego: 16 bajtów,
- 5 kluczy (tab. 6.36),
- warunek zatrzymania: sprawdzenie 1 600 000 000 permutacji lub $f_{fit} = 100\%$,
- losowa inicjacja początkowej permutacji,
- wartości parametrów zgodne z wynikami eksperymentu z pkt. 6.2.3:
  - funkcja dopasowania: dopasowanie ważone (wzór (4.8), str. 73),
  - $\tau_0 = 0{,}1$,
  - $\beta = 7$,
  - $q_0 = 0{,}1$,

  - $\rho = 0{,}7$,

- liczba niezależnych testów dla jednego zestawu parametrów: 32.

Łącznie przeprowadzono 160 testów ($5 \cdot 1 \cdot 32$).

Do badań samodzielnie wygenerowano wektory testowe tak, by zapewnić różnorodność analizowanych przypadków. Wartości kluczy użytych w badaniach umieszczono w tab. 6.36.

Z przeprowadzonych 160 testów 13 (8,1%) zakończyło się odnalezieniem prawidłowego stanu wewnętrznego w zadanej liczbie iteracji. W przypadku testów, które zakończyły się sukcesem, tj. odnalezieniem poszukiwanej permutacji, potrzebne było sprawdzenie średnio 573 620 889 permutacji przed odnalezieniem permutacji poszukiwanej. Stanowi to 0,0027% całej przestrzeni przeszukiwań. Jeden test trwał około 22 minuty i 56 sekund.

Wniosek

Przy użyciu algorytmu mrowiskowego uzyskano w zadanej liczbie sprawdzonych permutacji 10 razy mniej sukcesów niż przy użyciu przeszukiwania z tabu (6.2.4). Co więcej, testy, które zakończyły się sukcesem, potrzebowały sprawdzenia 2 razy większej liczby permutacji w przypadku algorytmu mrowiskowego w porównaniu do przeszukiwania z tabu. Zatem proponowana metoda kryptoanalizy przy użyciu algorytmu mrowiskowego nie wydaje się być skuteczna dla szyfru VMPC o rozmiarze 16.

6.2.6 Algorytm przeszukiwania z tabu w kryptoanalizie szyfru VMPC

W eksperymencie przeprowadzono proponowany atak kryptoanalityczny na polski szyfr VMPC. Następnie otrzymane wyniki porównano z istniejącymi atakami na ten szyfr.

Warunki przeprowadzenia eksperymentu były następujące:

- badany algorytm: przeszukiwanie z tabu (alg. 4.1),
- algorytm szyfrujący poddany kryptoanalizie: VMPC,
- długość analizowanego strumienia szyfrującego: 256 bajtów,
- 10 kluczy (tab. 6.38), tylko jeden wektor testowy został udostępniony przez autora [Żół04],
- warunek zatrzymania: 306 iteracji,
- losowa inicjacja początkowej permutacji,
- parametry dobrane zgodnie z wynikami eksperymentu z pkt. 6.2.1 (tab. 6.37),
- liczba niezależnych testów dla jednego zestawu parametrów: 32.

Łącznie przeprowadzono 320 testów ($10 \cdot 1 \cdot 32$).

Liczbę iteracji dobrano tak, by liczba przeanalizowanych rozwiązań w czasie jednego testu przeszukiwania z tabu była taka sama, jak dla kryptoanalizy RC4 (pkt 6.1.6). Ponieważ w trakcie jednego wykonania przeszukiwania z tabu dla rozpatrywanego rodzaju otoczenia („połowa par") analizowanych jest 16 320 rozwiązań, przyjęto liczbę iteracji równą 306 (306 iteracji analizujących po 16 320 rozwiązań to 4 993 920 rozpatrzonych rozwiązań).

Wygenerowane losowo 5 000 000 permutacji oceniono za pomocą tej samej funkcji dopasowania. Jest to w przybliżeniu równe liczbie permutacji ocenionych w trakcie jednego testu przeszukiwania z tabu.

Tabela 6.38: Klucze użyte w kryptoanalizie szyfru VMPC (pierwszy jest oficjalnym wektorem testowym [Żół04])

ID	Klucz	IV
2-0	0x9661410ab797d8a9eb767c21172df6c7	0x4b5c2f003e67f39557a8d26f3da2b155
2-1	0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	0x4b5c2f003e67f39557a8d26f3da2b155
2-2	0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	0x55555555555555555555555555555555
2-3	0x00000000000000000000000000000000	0x00000000000000000000000000000000
2-4	0x80000000000000000000000000000000	0x00000000000000000000000000000000
2-5	0x09090909090909090909090909090909	0x00000000000000000000000000000000
2-6	0x000102030405060708090a0b0c0d0e0f	0x00000000000000000000000000000000
2-7	0x288ff65dc42b92f960c70f62b5085bae	0x00000000000000000000000000000000
2-8	0x00000000000000000000000000000000	0x00000100000000000000000000000000
2-9	0x00000000000000000000000000000000	0x288ff65dc42b92f960c70f62b5085bae

Dla tego rodzaju otoczenia („połowa par") oraz permutacji o wielkości 256 bajtów rozmiar potencjalnego otoczenia to $\binom{256}{2} = 32\,640$, a rozmiar otoczenia analizowanego w danej iteracji: $16\,320$ (analizowana jest połowa możliwych par). Przy tym rozmiarze permutacji horyzont $3|S| = 768$ (gdzie $|S|$ to rozmiar permutacji).

Wyniki eksperymentu zostały zebrane w tab. 6.39. Znaczenie poszczególnych elementów tabeli jest takie samo jak dla tab. 6.19 w punkcie 6.1.6 (str. 110). Rezultaty dla przedstawionego zestawu parametrów to wartość funkcji dopasowania średnio 9,8% dla dopasowania bajtowego. Rezultaty są porównywalne dla różnych kluczy, więc nie zależą od szczególnych właściwości klucza. Otrzymane rezultaty są również dość stabilne i dobrze skupione, ponieważ odchylenie standardowe jest niewielkie ($< 2,0\%$). Strumień szyfrujący wygenerowany przez losowe permutacje osiągnął wartość funkcji dopasowania maksymalnie 3,9% (średnio 0,4%), co jest wartością dużo mniejszą niż uzyskana za pomocą przeszukiwania z tabu. Jeden test trwał około 18,4 sekundy.

Aproksymację przebiegu funkcji dopasowania sprawdzono za pomocą analizy regresji. Sprawdzono cztery typy regresji: liniową, logarytmiczną, wykładniczą i potęgową. Regresja

Tabela 6.39: Wyniki uzyskane dla szyfru VMPC; w przypadku algorytmu przeszukiwania z tabu dla każdego klucza podano wartość funkcji dopasowania oraz (poniżej) numer iteracji, w której tę wartość uzyskano. Dla permutacji wygenerowanych losowo dla każdego klucza podano uzyskaną wartość funkcji dopasowania w podanej liczbie prób. Użyte oznaczenia: min – wartość minimalna, $śr.$ – średnia arytmetyczna, max – wartość maksymalna, s – odchylenie standardowe

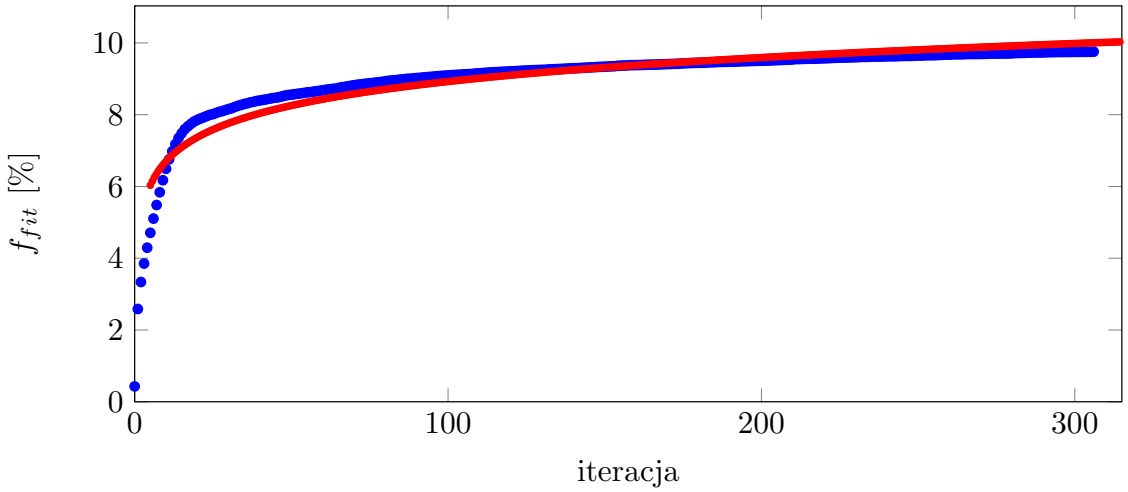
Dopasowanie	bajtowe							
Algorytm	przeszukiwanie z tabu				losowo			
ID klucza	min	$śr.$	max	s	min	$śr.$	max	s
2-0	7,4%	9,5%	11,7%	1,1%	0,0%	0,4%	3,9%	0,4%
	39	159	292	81	–	–	–	–
2-1	6,6%	9,7%	11,7%	1,3%	0,0%	0,4%	3,5%	0,4%
	21	152	285	86	–	–	–	–
2-2	6,6%	9,7%	12,5%	1,4%	0,0%	0,4%	3,5%	0,4%
	21	148	282	80	–	–	–	–
2-3	6,6%	9,7%	12,5%	1,2%	0,0%	0,4%	3,9%	0,4%
	23	156	306	84	–	–	–	–
2-4	7,0%	9,8%	12,1%	1,4%	0,0%	0,4%	3,5%	0,4%
	31	156	291	66	–	–	–	–
2-5	7,8%	9,8%	13,7%	1,3%	0,0%	0,4%	3,5%	0,4%
	24	187	305	90	–	–	–	–
2-6	6,3%	9,7%	14,1%	1,9%	0,0%	0,4%	3,5%	0,4%
	23	168	306	82	–	–	–	–
2-7	8,2%	10,2%	12,1%	1,1%	0,0%	0,4%	3,5%	0,4%
	47	174	303	87	–	–	–	–
2-8	6,6%	9,4%	12,5%	1,6%	0,0%	0,4%	3,5%	0,4%
	47	141	292	71	–	–	–	–
2-9	5,9%	10,1%	13,3%	1,4%	0,0%	0,4%	3,5%	0,4%
	25	154	306	82	–	–	–	–
razem	5,9%	9,8%	14,1%	1,4%	0,0%	0,4%	3,9%	0,4%
	18	160	306	80	–	–	–	–

logarytmiczna daje najmniejszą wartość błędu R^2, dlatego w dalszej analizie wyniki przedstawiane będą za jej pomocą.

Uśredniony przebieg funkcji dopasowania w kolejnych iteracjach przedstawiono na rys. 6.4 kolorem niebieskim. Na wykresie zaznaczono również kolorem czerwonym linię regresji logarytmicznej odpowiadającej przedstawionemu przebiegowi funkcji dopasowania. Równanie regresji logarytmicznej dla tego wykresu można opisać wzorem:

$$f(x) = 0,0096496251 \ln(x) + 0,0447960899 \qquad (R^2 = 0,903), \tag{6.3}$$

gdzie R^2 to współczynnik determinacji. Korzystając z ekstrapolacji można na podstawie powyższego wzoru obliczyć, że pełne pokrycie strumienia szyfrującego uda się osiągnąć po około 2^{143} iteracjach. W każdej iteracji sprawdzanych jest $\approx 2^{14}$ rozwiązań. Zatem w celu odnalezienia stanu wewnętrznego trzeba sprawdzić około 2^{157} możliwości.



Rysunek 6.4: Wykres średniej wartości funkcji dopasowania w kolejnych iteracjach przeszukiwania z tabu (szyfr VMPC) – niebieskie punkty; wykres regresji logarytmicznej zaznaczono kolorem czerwonym

Wniosek

Choć liczba ta pozostaje poza perspektywą praktycznego złamania szyfru, jest to znacznie lepszy wynik niż najlepszy ze znanych ataków (przedstawiony w punkcie 2.4.2) wymagający przeciętnie 2^{260} operacji obliczeniowych. Ponadto dalsze dopracowanie algorytmu może przybliżyć do opracowania skutecznej metody kryptoanalizy szyfru VMPC. Pozostałe z zaprezentowanych w punkcie 2.4.2 ataków są atakami rozróżniającymi i, jak wspominano wcześniej, ataki tego typu nie pozwalają bezpośrednio na wydedukowanie niczego ani na temat klucza, ani na temat stanu wewnętrznego, ani na temat tekstu jawnego. Zatem kryptoanaliza z przeszukiwaniem z tabu prezentowana w niniejszej rozprawie jest najmocniejszym znanym atakiem dla szyfru VMPC.

6.3 Kryptoanaliza szyfru RC4+

W poniższych eksperymentach zbadano możliwość użycia zaproponowanych algorytmów w kryptoanalizie szyfru strumieniowego RC4+, będącego modyfikacją szyfru RC4.

6.3.1 Dobór wartości parametrów algorytmu dla przeszukiwania z tabu

Celem eksperymentu było sprawdzenie wpływu poszczególnych parametrów algorytmu na jakość otrzymywanych rozwiązań. Do badań wybrano mniejszą wersję szyfru, w której założono, że permutacja przechowywana w tablicy S składa się z dziesięciu liczb z zakresu od 0 do 9. Wersję tę oznaczono jako RC4+_10. Zasada działania algorytmu jest taka sama jak w alg. 2.3, z tą różnicą, że wszystkie operacje wykonywane są $\mathrm{mod}\ 10$.

Warunki przeprowadzenia eksperymentu oraz jego scenariusz są takie same jak w przypadku użycia przeszukiwania z tabu do kryptoanalizy szyfru RC4_10 (pkt 6.1.1, str. 93). Jedyną różnicą jest to, że szyfrem poddanym kryptoanalizie w obecnym eksperymencie jest szyfr RC4+_10. W związku z tym założenia nie będą tutaj szczegółowo prezentowane. Przedstawione zostaną jedynie wyniki, które zaprezentowano w tab. 6.41-6.44. Jeden test trwał około 0,8 sekundy.

Tabela 6.40: Klucze użyte w kryptoanalizie szyfru RC4+_10

Klucz	IV
0x00070500	0x05020700
0x00000000	0x05020700
0x00000000	0x05050505
0x00000000	0x00000000
0x08000000	0x00000000
0x09090909	0x00000000
0x00010203	0x00000000
0x00030603	0x00000000
0x00000000	0x00000006
0x00000000	0x00030603

Tabela 6.41: Wartości funkcji dopasowania dla algorytmu przeszukiwania z tabu (szyfr RC4+_10)

Dopasowanie	Liczba sukcesów	
bajtowe	299	93,4%
bajtowe z sąsiadami	293	91,6%
ważone	**300**	**93,8%**
ważone odwrotnie	240	75,0%

Tabela 6.42: Wartości otoczenia dla algorytmu przeszukiwania z tabu (szyfr RC4+_10)

Otoczenie	Liczba sukcesów	
wszystkie pary	0	0,0%
połowa par	300	93,8%
elementy losowo	**314**	**98,1%**
elementy sąsiadująco	0	0,0%

Tabela 6.43: Wartości horyzontu dla algorytmu przeszukiwania z tabu (szyfr RC4+_10)

Horyzont	Liczba sukcesów	
$\ln \lvert R(r)\rvert$	310	96,9%
$\log_2 \lvert R(r)\rvert$	**316**	**98,8%**
$\sqrt{\lvert R(r)\rvert}$	314	98,1%
$\lvert R(r)\rvert/2$	308	96,3%
$3\lvert S\rvert$	311	97,2%
losowy	315	98,4%

Tabela 6.44: Wartości kryterium aspiracji dla algorytmu przeszukiwania z tabu (szyfr RC4+_10)

Aspiracja	Liczba sukcesów	
brak	**316**	**98,8%**
najlepszego	310	96,9%

Wniosek

Na podstawie tego eksperymentu na szyfrze RC4+_10 dobrano dla przeszukiwania z tabu następujący zestaw parametrów:

- funkcja dopasowania: dopasowanie ważone (wzór (4.8), str. 73),
- otoczenie: elementy losowo,
- horyzont: $\log_2 |R(r)|$ (gdzie $|R(r)|$ to rozmiar otoczenia rozwiązania r),
- kryterium aspiracji: brak.

Dla tych wartości parametrów poszukiwaną permutację znaleziono w 316 próbach z 320 (98,8%) po sprawdzeniu średnio 682 204 permutacji, co jest równoważne przeszukaniu 18,8% całej przestrzeni rozwiązań. Te wartości parametrów użyto w kolejnych eksperymentach dotyczących kryptoanalizy szyfru RC4+ przy wykorzystaniu algorytmu przeszukiwania z tabu (pkt. 6.3.4, 6.3.6).

6.3.2 Dobór wartości parametrów dla algorytmu mrówki wierzchołkowej

Celem eksperymentu było sprawdzenie algorytmu mrówki wierzchołkowej w kryptoanalizie szyfru RC4+_10 oraz dobór optymalnych parametrów algorytmu. Warunki przeprowadzenia eksperymentu oraz jego scenariusz są analogiczne jak w przypadku użycia algorytmu mrówki wierzchołkowej do kryptoanalizy szyfru RC4_10 (pkt 6.1.2, str. 98). Jedyną różnicą jest to, że szyfrem poddanym kryptoanalizie w obecnym eksperymencie jest szyfr RC4+_10. W związku z tym założenia nie będą tutaj szczegółowo prezentowane. Przedstawione zostaną jedynie wyniki, które zaprezentowano w tab. 6.45-6.49. Wartości kluczy użytych w badaniach umieszczono w tab. 6.40. Jeden test trwał około 6,2 sekundy.

Tabela 6.45: Wartości funkcji dopasowania dla algorytmu mrówki wierzchołkowej (szyfr RC4+_10)

Dopasowanie	Liczba sukcesów	
bajtowe	173	54,1%
bajtowe z sąsiadami	**201**	**62,8%**
ważone	183	57,2%
ważone odwrotnie	197	61,6%

Tabela 6.46: Wartości parametru τ_{max} dla algorytmu mrówki wierzchołkowej (szyfr RC4+_10)

τ_{max}	Liczba sukcesów	
$2\|S\|$	**201**	**62,8%**
$4\|S\|$	168	52,5%
∞ (brak)	0	0,0%

Tabela 6.47: Wartości parametru τ_0 dla algorytmu mrówki wierzchołkowej (szyfr RC4+_10)

τ_0	Liczba sukcesów	
1	**201**	**62,8%**
$\tau_{max}/2$	196	61,3%
τ_{max}	200	62,5%

Tabela 6.48: Wartości parametru ρ dla algorytmu mrówki wierzchołkowej (szyfr RC4+_10)

ρ	Liczba sukcesów	
1	**201**	**62,8%**
$\|S\|/2$	162	50,6%
$\|S\|$	151	47,2%

Tabela 6.49: Wartości budowy rozwiązań dla algorytmu mrówki wierzchołkowej (szyfr RC4+_10)

Budowa rozwiązań	Liczba sukcesów	
od początku	201	62,8%
od końca	**206**	64,4%
losowo	196	61,3%

Wniosek

Najlepsze wyniki algorytmu mrówki wierzchołkowej dla szyfru RC4+_10 osiągnięto dla następujących wartości parametrów:

- funkcja dopasowania: dopasowanie bajtowe z sąsiadami (wzór (4.6), str. 72),
- $\tau_{max} = 2|S|$ (gdzie $|S|$ to rozmiar permutacji),
- $\tau_0 = 1$,
- $\rho = 1$,
- budowa rozwiązań: od końca.

Przy takich ustawieniach 64,4% testów zakończyło się sukcesem. Wynik ten jest znacznie niższy niż podczas użycia przeszukiwania z tabu (pkt 6.3.1), gdzie osiągnięto 98,8% sukcesów. Na podstawie tych badań odrzucono możliwość efektywnego wykorzystania algorytmu mrówki wierzchołkowej w kryptoanalizie szyfru RC4+.

6.3.3 Dobór wartości parametrów dla algorytmu mrowiskowego

Celem eksperymentu było sprawdzenie algorytmu mrowiskowego w kryptoanalizie szyfru RC4+_10 oraz dobór optymalnych parametrów algorytmu. Warunki przeprowadzenia eksperymentu oraz jego scenariusz są takie same jak w przypadku użycia algorytmu mrowiskowego do kryptoanalizy szyfru RC4_10 (pkt 6.1.3, str. 101). Jedyną różnicą jest to, że szyfrem poddanym kryptoanalizie w aktualnym eksperymencie jest szyfr RC4+_10. W związku z tym założenia nie będą tutaj szczegółowo prezentowane. Przedstawione zostaną jedynie wyniki, które zaprezentowano w tab. 6.50-6.54. Wartości kluczy użytych w badaniach umieszczono w tab. 6.40. Jeden test trwał około 1,3 sekundy.

Wniosek

Podsumowując, w wyniku dostrajania parametrów algorytmu mrowiskowego dla szyfru RC4+_10 ustalono następujące ich wartości:

- funkcja dopasowania: dopasowanie ważone (wzór (4.8), str. 73),

Tabela 6.50: Wartości funkcji dopasowania dla algorytmu mrowiskowego (szyfr RC4+_10)

Dopasowanie	Liczba sukcesów	
bajtowe	260	81,3%
bajtowe z sąsiadami	243	75,9%
ważone	**266**	**83,1%**
ważone odwrotnie	232	72,5%

Tabela 6.51: Wartości parametru τ_0 dla algorytmu mrowiskowego (szyfr RC4+_10)

τ_0	Liczba sukcesów	
0,1	**282**	**88,1%**
0,01	264	82,5%
0,001	269	84,1%
0,0001	266	83,1%
0,00001	267	83,4%

Tabela 6.52: Wartości parametru β dla algorytmu mrowiskowego (szyfr RC4+_10)

β	Liczba sukcesów	
0,5	251	78,4%
1	264	82,5%
3	302	94,4%
5	304	95,0%
7	**312**	**97,5%**
9	309	96,6%
15	310	96,9%

Tabela 6.53: Wartości parametru q_0 dla algorytmu mrowiskowego (szyfr RC4+_10)

q_0	Liczba sukcesów	
0,1	**312**	**97,5%**
0,3	303	94,7%
0,5	286	89,4%
0,7	253	79,1%
0,9	200	62,5%

Tabela 6.54: Wartości parametru ρ dla algorytmu mrowiskowego (szyfr RC4+_10)

ρ	Liczba sukcesów	
0,1	308	96,3%
0,3	309	96,6%
0,5	**315**	**98,4%**
0,7	307	95,9%
0,9	310	96,99%

- τ_0 = 0,1,
- β = 7,
- q_0 = 0,1,
- ρ = 0,5.

Dla tych wartości parametrów poszukiwaną permutację znaleziono w 315 z 320 prób (98,4%). W przypadku testów, które zakończyły się sukcesem, potrzebne było sprawdzenie średnio 704 638 permutacji przed odnalezieniem tej prawidłowej. Stanowi to 19,4% całej przestrzeni poszukiwań. Tych wartości parametrów użyto w kolejnych eksperymentach dotyczących kryptoanalizy szyfru RC4+ przy wykorzystaniu algorytmu mrowiskowego (pkt 6.3.5).

6.3.4 Algorytm przeszukiwania z tabu w kryptoanalizie szyfru RC4+_16

Celem eksperymentu było sprawdzenie możliwości kryptoanalizy proponowaną w rozprawie metodą większej niż w poprzednich eksperymentach wersji szyfru RC4+. Do badań wybrano wersję szyfru, w której założono, że permutacja przechowywana w tablicy S składa się z szesnastu liczb z zakresu od 0 do 15. Oznaczono tę wersję jako RC4+_16. Zasada działania szyfru jest taka sama jak w alg. 2.3, z jedyną różnicą taką, że wszystkie operacje wykonywane są $\mathrm{mod}\ 16$. Liczba możliwych kluczy dla tej wielkości szyfru RC4+ to 2^{128}, a liczba możliwych początkowych stanów wewnętrznych to $16! = 20\,922\,789\,888\,000 \approx 2^{44}$.

Warunki przeprowadzenia eksperymentu były następujące:

- badany algorytm: przeszukiwanie z tabu (alg. 4.1),
- algorytm szyfrujący poddany kryptoanalizie: RC4+_16,
- długość analizowanego strumienia szyfrującego: 16 bajtów,
- 5 kluczy (tab. 6.55),
- warunek zatrzymania: sprawdzenie 1 600 000 000 permutacji lub $f_{fit} = 100\%$,
- losowa inicjacja początkowej permutacji,
- parametry dobrane zgodnie z wynikami eksperymentu z pkt. 6.3.1 (tab. 6.56),
- liczba niezależnych testów dla jednego zestawu parametrów: 32.

Tabela 6.55: Klucze użyte w kryptoanalizie szyfru RC4+_16

Klucz	IV
0x0A0A0A0A	0x0B0C0F00
0x0A0A0A0A	0x05050505
0x08000000	0x00000000
0x00010203	0x00000000
0x080F060D	0x0B0C0F0D

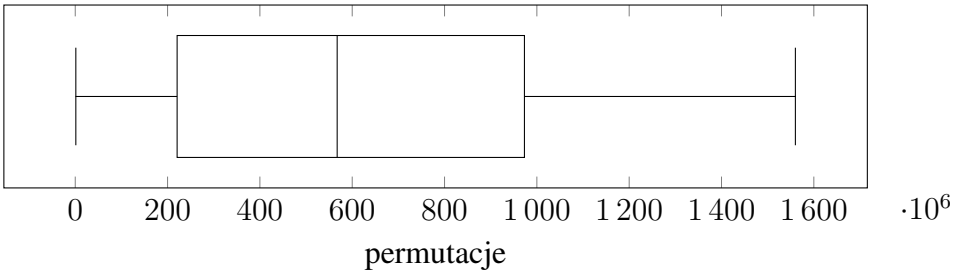
Tabela 6.56: Wartości parametrów użyte w kryptoanalizie szyfru RC4+_16

Dopasowanie	Horyzont	Otoczenie	Kryterium aspiracji
ważone	$\log_2 \lvert R(r) \rvert = 7$	elementy losowo	brak

Łącznie przeprowadzono 160 testów ($5 \cdot 1 \cdot 32$).

Dla tego rodzaju otoczenia („elementy losowo”) oraz permutacji o wielkości 16 bajtów rozmiar potencjalnego otoczenia to $\binom{16}{2} = 120$, a rozmiar otoczenia analizowanego w danej iteracji: 16.

Z przeprowadzonych 160 testów 124 (77,5%) zakończyło się odnalezieniem prawidłowej permutacji w zadanej liczbie sprawdzonych permutacji. Na rys. 6.5 przedstawiono w formie wykresu pudełkowego liczbę permutacji sprawdzonych przed znalezieniem permutacji poszukiwanej. Na wykresie uwzględniono tylko te testy, które zakończyły się sukcesem, tj. odnaleziono poszukiwaną permutację. Średnio potrzebne było sprawdzenie 642 796 960 permutacji przed odnalezieniem permutacji poszukiwanej. Stanowi to 0,0031% całej przestrzeni rozwiązań. Jeden test trwał około 31 minut i 48 sekund.



Rysunek 6.5: Liczba permutacji potrzebnych do znalezienia prawidłowej permutacji szyfru RC4+_16

Wniosek

Proponowana metoda kryptoanalizy przy użyciu przeszukiwania z tabu jest skuteczna dla szyfru RC4+ o rozmiarze 16.

6.3.5 Algorytm mrowiskowy w kryptoanalizie szyfru RC4+_16

Celem eksperymentu było sprawdzenie możliwości kryptoanalizy proponowanym w rozprawie algorytmem mrowiskowym większej niż poprzednio (pkt 6.3.3) wersji szyfru RC4+. Warunki przeprowadzenia eksperymentu były następujące:

- badany algorytm: algorytm mrowiskowy (alg. 5.2),
- algorytm szyfrujący poddany kryptoanalizie: RC4+_16,
- długość analizowanego strumienia szyfrującego: 16 bajtów,
- 5 kluczy (tab. 6.55),
- warunek zatrzymania: sprawdzenie 1 600 000 000 permutacji lub $f_{fit} = 100\%$,
- losowa inicjacja początkowej permutacji,
- wartości parametrów zgodne z wynikami eksperymentu z pkt. 6.3.3:
  - funkcja dopasowania: dopasowanie ważone (wzór (4.8), str. 73),
  - τ_0 = 0,1,
  - β = 7,
  - q_0 = 0,1,
  - ρ = 0,5,
- liczba niezależnych testów dla jednego zestawu parametrów: 32.

Łącznie przeprowadzono 160 testów ($5 \cdot 1 \cdot 32$).

Z przeprowadzonych 160 testów 10 (6,3%) zakończyło się odnalezieniem prawidłowego stanu wewnętrznego w zadanej liczbie iteracji. W przypadku testów, które zakończyły się sukcesem, tj. odnalezieniem poszukiwanej permutacji, potrzebne było sprawdzenie średnio 658 149 361 permutacji przed odnalezieniem permutacji poszukiwanej. Stanowi to 0,0031% całej przestrzeni przeszukiwań. Jeden test trwał około 41 minut i 13 sekund.

Wniosek

Testy, które zakończyły się sukcesem, potrzebowały sprawdzenia porównywalnej liczby permutacji w przypadku algorytmu mrowiskowego oraz przeszukiwania z tabu. Jednak przy użyciu algorytmu mrowiskowego uzyskano w zadanej liczbie sprawdzonych permutacji 12 razy mniej sukcesów niż przy użyciu przeszukiwania z tabu (pkt 6.3.4). Zatem proponowana metoda kryptoanalizy przy użyciu algorytmu mrowiskowego nie wydaje się być skuteczna dla szyfru RC4+ o rozmiarze 16.

6.3.6 Algorytm przeszukiwania z tabu w kryptoanalizie szyfru RC4+

Badanie wykonano w celu pokazania możliwości zaproponowanego ataku na szyfr RC4+. Wyniki osiągnięte przez kryptoanalizę z przeszukiwaniem z tabu zestawiono również ze znanymi obecnie atakami na ten szyfr.

Warunki przeprowadzenia eksperymentu były następujące:

- badany algorytm: przeszukiwanie z tabu (alg. 4.1),
- algorytm szyfrujący poddany kryptoanalizie: RC4+,
- długość analizowanego strumienia szyfrującego: 256 bajtów,
- 10 kluczy (tab. 6.57),
- warunek zatrzymania: 19531 iteracji,
- losowa inicjacja początkowego stanu wewnętrznego,
- 1 wybrany zestaw parametrów (tab. 6.56),
- liczba niezależnych testów dla jednego zestawu parametrów: 32.

Łącznie przeprowadzono 320 testów ($10 \cdot 1 \cdot 32$).

Liczbę iteracji dobrano tak, by liczba przeanalizowanych rozwiązań w czasie jednego testu przeszukiwania z tabu była taka sama, jak dla kryptoanalizy RC4 (pkt 6.1.6) oraz VMPC (pkt 6.2.6). Ponieważ w trakcie jednego wykonania przeszukiwania z tabu dla rozpatrywanego rodzaju otoczenia („połowa par") analizowanych jest 256 rozwiązań, przyjęto liczbę iteracji równą 19 531 (19 531 iteracji analizujących po 256 rozwiązań to 4 999 936 rozpatrzonych rozwiązań). Wygenerowane losowo 5 000 000 permutacji oceniono za pomocą tej samej funkcji dopasowania.

Dla tego rodzaju otoczenia („elementy losowo") oraz permutacji o wielkości 256 bajtów rozmiar potencjalnego otoczenia to $\binom{256}{2} = 32\,640$, a rozmiar otoczenia analizowanego w danej iteracji: 256. Przy tym rozmiarze permutacji horyzont $\log_2 |R(r)| = 151$ (gdzie $|R(r)|$ to rozmiar otoczenia rozwiązania r).

Wyniki eksperymentu zostały zebrane w tab. 6.58. Znaczenie poszczególnych elementów tabeli jest takie samo jak dla tab. 6.19 w punkcie 6.1.6 (str. 110). Uzyskano wartość funkcji dopasowania w wysokości 44,7% z dopasowaniem ważonym (prawidłowo odtworzono średnio 28,6% strumienia, czyli ok. 73 bajty). Rezultaty są porównywalne dla różnych kluczy, więc nie zależą od szczególnych właściwości klucza. Otrzymane rezultaty są również dość stabilne i dobrze skupione, ponieważ odchylenie standardowe jest niewielkie ($< 2,0\%$). Strumienie szyfrujące losowo wygenerowanych permutacji nie przekroczyły progu 4,9%

Tabela 6.57: Klucze użyte w kryptoanalizie szyfru RC4+

ID	Klucz	IV
3-0	0x9661410ab797d8a9eb767c21172df6c7	0x4b5c2f003e67f39557a8d26f3da2b155
3-1	0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	0x4b5c2f003e67f39557a8d26f3da2b155
3-2	0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	0x55555555555555555555555555555555
3-3	0x00000000000000000000000000000000	0x00000000000000000000000000000000
3-4	0x80000000000000000000000000000000	0x00000000000000000000000000000000
3-5	0x09090909090909090909090909090909	0x00000000000000000000000000000000
3-6	0x000102030405060708090a0b0c0d0e0f	0x00000000000000000000000000000000
3-7	0x288ff65dc42b92f960c70f62b5085bae	0x00000000000000000000000000000000
3-8	0x00000000000000000000000000000000	0x00000100000000000000000000000000
3-9	0x00000000000000000000000000000000	0x288ff65dc42b92f960c70f62b5085bae

wartości funkcji dopasowania (średnio 0,4%), co jest wartością dużo mniejszą niż uzyskana za pomocą przeszukiwania z tabu. Jeden test trwał około 1 minutę i 4 sekundy.

Aproksymację przebiegu funkcji dopasowania sprawdzono za pomocą analizy regresji. Sprawdzono cztery typy regresji: liniową, logarytmiczną, wykładniczą i potęgową. Regresja logarytmiczna daje najmniejszą wartość błędu R^2, dlatego w dalszej analizie wyniki przedstawiane będą za jej pomocą.

Na rys. 6.6 kolorem niebieskim oznaczono uśredniony przebieg funkcji dopasowania w kolejnych iteracjach. Kolorem czerwonym oznaczono linię regresji logarytmicznej odpowiadającej przedstawionemu wykresowi przebiegu funkcji dopasowania. Stąd można określić następujące równanie regresji logarytmicznej:

$$f(x) = 0,0277065555 \ln(x) + 0,1469160456 \qquad (R^2 = 0,904), \tag{6.4}$$

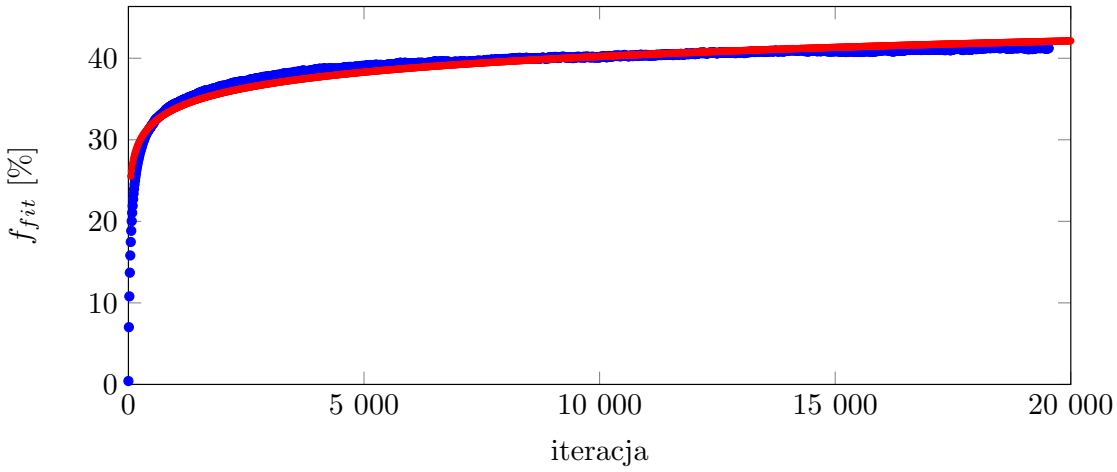
gdzie R^2 to współczynnik determinacji. Na podstawie tego równania można policzyć, że stan wewnętrzny zostanie znaleziony po około 2^{44} iteracjach (każda iteracja to sprawdzenie 2^8 stanów wewnętrznych). Stan wewnętrzny można zatem uzyskać po zweryfikowaniu około 2^{52} permutacji początkowych tablicy S.

Wniosek

Aktualnie nie jest znany atak na szyfr RC4+ umożliwiający odszyfrowanie tajnej wiadomości, bez ingerencji w stan wewnętrzny algorytmu (pkt 2.4.3). Przedstawiona metoda kryptoanalizy z przeszukiwaniem z tabu jest zatem pierwszym atakiem na szyfr RC4+, który może doprowadzić do złamania algorytmu.

Tabela 6.58: Wyniki uzyskane dla szyfru RC4+; w przypadku algorytmu przeszukiwania z tabu dla każdego klucza podano wartość funkcji dopasowania oraz (poniżej) numer iteracji, w której tę wartość uzyskano. Dla permutacji wygenerowanych losowo dla każdego klucza podano uzyskaną wartość funkcji dopasowania w podanej liczbie prób. Użyte oznaczenia: min – wartość minimalna, $śr.$ – średnia arytmetyczna, max – wartość maksymalna, s – odchylenie standardowe

Dopasowanie	ważone							
Algorytm	przeszukiwanie z tabu				losowo			
ID klucza	min	$śr.$	max	s	min	$śr.$	max	s
3-0	40,3%	44,3%	47,3%	1,6%	0,0%	0,4%	4,7%	0,4%
	4 416	13 158	18 829	4 041	–	–	–	–
3-1	42,3%	44,9%	47,3%	1,1%	0,0%	0,4%	4,4%	0,4%
	3 618	14 693	19 473	3 767	–	–	–	–
3-2	40,7%	44,9%	47,6%	1,5%	0,0%	0,4%	4,5%	0,4%
	6 077	15 333	19 463	3 770	–	–	–	–
3-3	42,5%	44,7%	49,0%	1,5%	0,0%	0,4%	4,4%	0,4%
	5 338	14 669	19 475	3 626	–	–	–	–
3-4	40,8%	44,8%	48,1%	1,7%	0,0%	0,4%	4,9%	0,4%
	4 531	14 745	19 525	4 203	–	–	–	–
3-5	42,1%	44,4%	46,6%	1,2%	0,0%	0,4%	4,4%	0,4%
	8 306	14 653	19 528	3 295	–	–	–	–
3-6	41,4%	44,5%	48,5%	1,6%	0,0%	0,4%	4,4%	0,4%
	5 216	14 979	19 345	3 807	–	–	–	–
3-7	41,9%	44,6%	48,5%	1,5%	0,0%	0,4%	4,8%	0,5%
	5 212	14 906	19 508	4 003	–	–	–	–
3-8	42,4%	44,8%	48,4%	1,4%	0,0%	0,4%	4,5%	0,4%
	7 150	15 511	19 449	3 613	–	–	–	–
3-9	42,5%	44,9%	47,5%	1,4%	0,0%	0,4%	4,9%	0,4%
	4 856	15 052	19 530	4 088	–	–	–	–
razem	40,3%	44,7%	49,0%	1,5%	0,0%	0,4%	4,9%	0,4%
	3 618	14 770	19 530	3 824	–	–	–	–



Rysunek 6.6: Wykres średniej wartości funkcji dopasowania w kolejnych iteracjach przeszukiwania z tabu (szyfr RC4+) – niebieskie punkty; wykres regresji logarytmicznej zaznaczono kolorem czerwonym

6.4 Prawdopodobieństwo przypadkowej zgodności

Wyniki z poprzednich podrozdziałów zostaną porównane z wynikami przypadkowej zgodności ciągu 256 liczb z przedziału $\langle 0, 255 \rangle$, czyli z próbą losowego zgadnięcia strumienia szyfrującego. Niech p oznacza prawdopodobieństwo sukcesu, to znaczy że losowana lub zgadywana liczba jest taka sama jak kolejna liczba w ciągu (strumieniu szyfrującym). Niech $1 - p$ oznacza prawdopodobieństwo porażki, tzn. że losowana lub zgadywana liczba jest różna od tej, która występuje w ciągu (strumieniu szyfrującym). Prawdopodobieństwo k sukcesów w n próbach może być obliczone ze schematu Bernoulliego:

$$\binom{n}{k} p^k (1-p)^{n-k}. \tag{6.5}$$

W omawianym przypadku prawdopodobieństwo pojedynczego sukcesu $p = \frac{1}{256}$, a liczba prób $n = 256$. Zatem powyższy wzór przyjmie postać:

$$\binom{256}{k} \left(\frac{1}{256}\right)^k \left(\frac{255}{256}\right)^{256-k}. \tag{6.6}$$

Prawdopodobieństwa wybranej liczby sukcesów (liczb zgodnych z analizowanym ciągiem) zostały umieszczone w tab. 6.59.

Łączne prawdopodobieństwo zgodności poniżej 5 liczb wynosi 0,9964. Łączne prawdopodobieństwo prawidłowej zgodności do 10 liczb wynosi 0,999999992. Zatem prawdopodobieństwo przypadkowej zgodności więcej niż 10 liczb wynosi $0{,}000000008 \approx 2^{-27}$. Atak zaprezentowany w niniejszej rozprawie po przeanalizowaniu około 5 000 000 permutacji pozwolił na prawidłowe odtworzenie 25 bajtów (9,8%) dla

Tabela 6.59: Prawdopodobieństwo przypadkowej zgodności kolejnych liczb 256-elementowego ciągu

Zgodnych liczb k	Procent zgodnych liczb	Prawdopodobieństwo	
0	0,0%	0,3672	$\approx 2^{-1,4}$
1	0,4%	0,3686	$\approx 2^{-1,4}$
2	0,8%	0,1843	$\approx 2^{-2,4}$
3	1,2%	0,0612	$\approx 2^{-4}$
4	1,6%	0,0152	$\approx 2^{-6}$
5	2,0%	0,0030	$\approx 2^{-8}$
6	2,4%	0,0005	$\approx 2^{-11}$
7	2,7%	$6,89 \cdot 10^{-5}$	$\approx 2^{-14}$
8	3,1%	$8,41 \cdot 10^{-6}$	$\approx 2^{-17}$
9	3,5%	$9,09 \cdot 10^{-7}$	$\approx 2^{-20}$
10	3,9%	$8,81 \cdot 10^{-8}$	$\approx 2^{-23}$
⋮			
25	9,8%	$7,78 \cdot 10^{-27}$	$\approx 2^{-87}$
⋮			
69	26,9%	$1,14 \cdot 10^{-103}$	$\approx 2^{-342}$
⋮			
73	28,5%	$1,22 \cdot 10^{-111}$	$\approx 2^{-368}$
⋮			
256	100,0%	$3,09 \cdot 10^{-617}$	$= 2^{-2048}$

szyfru VMPC, 69 bajtów (26,9%) dla szyfru RC4 oraz 73 bajtów (28,5%) dla szyfru RC4+. Jak widać ze schematu Bernoulliego przypadkowa zgodność tylu liczb (bajtów) jest bardzo mało prawdopodobna. Co więcej, przypadkowa zgodność całej serii 256 liczb jest równa $3,09 \cdot 10^{-617} = 2^{-2048}$, czyli zaniedbywalnie mało.

Wniosek

To potwierdza, że permutacja, która wygeneruje taki analizowany ciąg (strumień szyfrujący), niemal na pewno będzie właściwą (poszukiwaną) permutacją.

6.5 Wnioski z badań eksperymentalnych

W ramach eksperymentów zbadano możliwość użycia algorytmu przeszukiwania z tabu, algorytmu mrówki wierzchołkowej oraz algorytmu mrowiskowego do rozwiązania problemu kryptoanalizy szyfrów strumieniowych zbudowanych z permutacji. Najpierw sprawdzono wszystkie proponowane algorytmy na mniejszych, dziesięcioelementowych

wersjach szyfrów strumieniowych RC4 (pkt. 6.1.1-3), VMPC (pkt. 6.2.1-3) i RC4+ (pkt. 6.3.1-3). Te eksperymenty posłużyły zarówno do dobrania najlepszych wartości parametrów, jak i odrzucenia algorytmu dającego najgorsze wyniki.

Pierwszym sprawdzanym parametrem w przypadku każdego algorytmu był rodzaj funkcji dopasowania. W rozprawie sprawdzono cztery rodzaje funkcji dopasowania, z czego trzy autorskie (bajtowe z sąsiadami, ważone, ważone odwrotnie) (podrozdz. 4.2). Dla przeszukiwania z tabu najlepsze wyniki uzyskano przy użyciu dopasowania bajtowego przy szyfrach RC4 (tab. 6.2) oraz VMPC (tab. 6.22). Dopasowanie bajtowe z sąsiadami pozwoliło uzyskać wyniki niewiele gorsze. Wyraźnie gorzej sprawdziły się dopasowanie ważone oraz ważone odwrotnie. Odmienna sytuacja miała miejsce w przypadku szyfru RC4+ (tab. 6.41). Tutaj najlepsze wyniki uzyskano dla dopasowania ważonego. Dopasowanie bajtowe oraz bajtowe z sąsiadami pozwoliły uzyskać wyniki niewiele gorsze, natomiast wyraźnie gorsze wyniki uzyskano używając dopasowania ważonego odwrotnie.

Kolejnym analizowanym parametrem w przypadku przeszukiwania z tabu było otoczenie. W rozprawie zaproponowano i przeanalizowano cztery rodzaje otoczenia (podrozdz. 4.1). Dla wszystkich trzech szyfrów najlepsze wyniki uzyskano dla otoczenia „połowa par" oraz „elementy losowo". Wyniki były dość podobne, choć pierwszy z nich lepiej sprawdził się w przypadku szyfru VMPC (tab. 6.23), a drugi – szyfrów RC4 (tab. 6.3) i RC4+ (tab. 6.42). Pozostałe dwa rodzaje otoczenia najczęściej nie pozwoliły na odnalezienie poszukiwanej permutacji w żadnym teście. W przypadku otoczenia „wszystkie pary" rozmiar analizowanego otoczenia był największych ze wszystkich rozważanych, natomiast liczba faktycznych zamian – najmniejsza. Należy przypuszczać, że to było przyczyną porażki w tym przypadku. W przypadku otoczenia „elementy sąsiadująco" wydaje się, że z powodu ograniczonych możliwości ruchu algorytm utknął w optimum lokalnym, z którego nie umiał się wydostać.

Dla przeszukiwania z tabu analizowano w dalszej kolejności horyzont, który jest odpowiedzialny za liczbę iteracji, przez które dany ruch jest zabroniony. Dla każdego szyfru najlepszy okazał się inny horyzont, choć dla wszystkich trzech szyfrów była niewielka (do 4%) różnica skuteczności pomiędzy wartościami horyzontów (tab. 6.4, 6.24, 6.43).

Ostatnim parametrem dla przeszukiwania z tabu było kryterium aspiracji, które określa, czy można zaakceptować ruch będący na liście tabu, o ile prowadzi on do rozwiązania lepszego niż znalezione do tej pory. Dla wszystkich trzech szyfrów uzyskano większą skuteczność przy braku kryterium aspiracji, choć różnice nie były duże (poniżej 6%) (tab. 6.5, 6.25, 6.44).

Podsumowując, na wyniki otrzymane podczas użycia przeszukiwania z tabu:

- największy wpływ miały:
  - funkcja dopasowania (tab. 6.2, 6.22, 6.41),

  - otoczenie (tab. 6.3, 6.23, 6.42),

- niewielki wpływ miały:
  - horyzont (tab. 6.4, 6.24, 6.43),
  - kryterium aspiracji (tab. 6.5, 6.25, 6.44).

Dla algorytmu mrówki wierzchołkowej funkcja dopasowania miała istotny wpływ (powyżej 8%) na osiągane wyniki w przypadku szyfrów RC4 (tab. 6.6) i RC4+ (tab. 6.45). Dla szyfru RC4 najlepsze było dopasowanie ważone odwrotnie, dla szyfru RC4+ – bajtowe z sąsiadami. Wyraźnie gorsze wyniki w przypadku obu tych szyfrów zaobserwowano dla dopasowania bajtowego i ważonego. W przypadku szyfru VMPC różnice między rodzajami funkcji dopasowania nie były aż tak istotne (poniżej 5%), a najlepiej sprawdziło się dopasowanie ważone (tab. 6.26).

Kolejnym parametrem w tym algorytmie był parametr τ_{max}, który odpowiada za ograniczenie akumulacji feromonu odkładanego przez mrówki na mapie feromonowej. Wybór tego parametru bardzo istotnie wpływał na wyniki (tab. 6.7, 6.27, 6.46). Dla wszystkich szyfrów najlepsze wyniki uzyskano przy wartości $\tau_{max} = 2|S|$. Znacznie gorsze rezultaty osiągnięto w przypadku $\tau_{max} = 4|S|$, a dla nieograniczonej akumulacji feromonu ($\tau_{max} = \infty$) niemal wszystkie testy zakończyły się niepowodzeniem. Prawdopodobnie w krótkim czasie następowała silna akumulacja feromonu na jednej ze ścieżek, która stanowiła optimum lokalne, a brak możliwości ograniczenia tej akumulacji tylko nasilał nasycenie tej ścieżki feromonem w wyniku dodatniego sprzężenia zwrotnego, pomimo mechanizmu odparowania.

Nie tak istotna była wartość feromonu domyślnego τ_0. Dla wszystkich trzech szyfrów różnica wyników była mała (poniżej 5%) (tab. 6.8, 6.28, 6.47). Dla szyfrów RC4 i VMPC najlepsze wyniki osiągnięto, gdy $\tau_0 = \tau_{max}$. W tej sytuacji w pierwszej fazie algorytmu następuje silna eksploracja przestrzeni rozwiązań. Dla szyfru RC4+ najlepsze wyniki osiągnięto, gdy $\tau_0 = 1$.

W dalszym toku eksperymentów sprawdzono parametr ρ odpowiedzialny za odparowanie feromonu. Najlepsze dla wszystkich szyfrów wyniki uzyskano, gdy $\rho = 1$ (tab. 6.9, 6.29, 6.48). Wartość tego parametru była istotna w przypadku szyfrów RC4 i RC4+ (różnice powyżej 15%), a mało istotna w przypadku szyfru VMPC (różnice poniżej 4%).

Ostatnim parametrem, w przypadku algorytmu mrówki wierzchołkowej, była budowa rozwiązań. W rozprawie zaproponowano trzy autorskie rodzaje budowy rozwiązań: od początku, od końca i losowo. Dla każdego z szyfrów najlepsze wyniki uzyskano dla innego rodzaju budowy rozwiązań: szyfr RC4 – budowa rozwiązań losowo (tab. 6.10), szyfr VMPC – budowa rozwiązań od początku (tab. 6.30), szyfr RC4+ – budowa rozwiązań od końca

(tab. 6.49). Ten parametr miał najbardziej istotny wpływ na wyniki dla szyfru VMPC (powyżej 10%), mały dla szyfru RC4 (poniżej 7%) i najmniejszy dla szyfru RC4+ (ok. 3%).

Podsumowując, na wyniki otrzymane podczas użycia algorytmu mrówki wierzchołkowej:

- największy wpływ miały:
  - funkcja dopasowania (w przypadku szyfrów RC4 i RC4+, tab. 6.6, 6.45),
  - parametr ρ (w przypadku szyfrów RC4 i RC4+, tab. 6.9, 6.48),
  - budowa rozwiązań (w przypadku szyfru VMPC, tab. 6.30),
  - parametr τ_{max} (tab. 6.7, 6.27, 6.46),
- niewielki wpływ miały:
  - funkcja dopasowania (w przypadku szyfru VMPC, tab. 6.26),
  - parametr ρ (w przypadku szyfru VMPC, tab. 6.29),
  - budowa rozwiązań (w przypadku szyfrów RC4 i RC4+, tab.6.10, 6.49),
  - parametr τ_0 (tab. 6.8, 6.28, 6.47).

W przypadku algorytmu mrowiskowego część parametrów miała duży, a część znikomy wpływ na zbieżność do poszukiwanego optimum globalnego. Z przeprowadzonych badań wynika, że najmniejszy wpływ miał parametr ρ, który jest odpowiedzialny za odparowanie śladu feromonowego. Dla wszystkich trzech szyfrów niezależnie od przyjętych wartości tego parametru wyniki były dość podobne (różnice poniżej 4%) (tab. 6.15, 6.35, 6.54). W przypadku każdego szyfru inna wartość okazała się najlepsza.

Parametr τ_0, który określa wartość feromonu domyślnego, miał niewielki wpływ na wyniki w przypadku wszystkich trzech szyfrów (różnice poniżej 6%) (tab. 6.12, 6.32, 6.51). Najlepsze okazały się najwyższe z przetestowanych wartości (0,1 lub 0,01).

Znaczne różnice w liczbie uzyskanych sukcesów są wyraźnie widoczne dla pozostałych parametrów, tj. parametru β oraz q_0, a także przyjętej funkcji dopasowania. Dla wszystkich trzech szyfrów najlepsze wyniki osiągnięto dla dopasowania ważonego, w drugiej kolejności dla dopasowania bajtowego. Najgorsze okazały się dopasowania bajtowe z sąsiadami oraz ważone odwrotnie (powyżej 9% względem najlepszego) (tab. 6.11, 6.31, 6.50). Zaproponowane w rozprawie dopasowanie ważone oraz bajtowe z sąsiadami pozwoliły na osiągnięcie większej liczby sukcesów niż dopasowanie bajtowe znane z literatury.

Dla wszystkich trzech szyfrów widoczna jest również zależność pomiędzy liczbą sukcesów a proporcjami informacji feromonowej i heurystycznej. W przypadku większej istotności informacji feromonowej ($\beta = 0,5$) liczba sukcesów była najmniejsza. Liczba

sukcesów wzrastała, gdy informacja feromonowa oraz heurystyczna były równie istotne ($\beta = 1$). Dalsza poprawa następowała, gdy informacja heurystyczna była bardziej istotna niż informacja feromonowa ($\beta = 3$). Jednak najlepsze wyniki (największą liczbę sukcesów) uzyskano w przypadku wyraźnej przewagi informacji heurystycznej ($\beta \geqslant 5$). Warto zauważyć, że dla wszystkich szyfrów najlepsza okazała się wartość $\beta = 7$. Gdy informacja heurystyczna była bardziej istotna osiągnięto od 8% dla szyfru RC4 do 20% dla szyfru VMPC więcej sukcesów (tab. 6.13, 6.33, 6.52) niż gdy bardziej istotny był ślad feromonowy.

Ostatnim parametrem, który miał istotny wpływ (powyżej 12%) na zbieżność, jest parametr q_0 sterujący zależnością pomiędzy eksploracją a eksploatacją. Dla wszystkich analizowanych szyfrów najlepszą wartością dla tego parametru okazała się wartość $q_0 = 0,1$. Im większa jego wartość, tym mniejsza liczba sukcesów (tab. 6.14, 6.34, 6.53). Małe q_0 oznacza większy nacisk na eksplorację przestrzeni rozwiązań.

Podsumowując, na wyniki otrzymane podczas użycia algorytmu mrowiskowego:

- największy wpływ miały:
  - funkcja dopasowania (tab. 6.11, 6.31, 6.50),
  - parametr β (tab. 6.13, 6.33, 6.52),
  - parametr q_0 (tab. 6.14, 6.34, 6.53),
- niewielki wpływ miały:
  - parametr τ_0 (tab. 6.12, 6.32, 6.51),
  - parametr ρ (tab. 6.15, 6.35, 6.54).

Eksperymenty na dziesięcioelementowych wersjach szyfrów posłużyły również do odrzucenia algorytmu dającego najgorsze wyniki. Dla wszystkich trzech szyfrów był to algorytm mrówki wierzchołkowej. Zbiorcze wyniki zaprezentowano w tab. 6.60. Podczas gdy dla przeszukiwania z tabu oraz algorytmu mrowiskowego skuteczność nie spadła poniżej 95%, algorytm mrówki wierzchołkowej nie osiągnął więcej niż 77% skuteczności.

Następnie pozostałe dwa algorytmy, czyli przeszukiwanie z tabu oraz algorytm mrowiskowy, zastosowano do kryptoanalizy wersji szesnastoelementowej tych samych szyfrów strumieniowych (pkt. 6.1.4-5, 6.2.4-5, 6.3.4-5). Zbiorcze wyniki zaprezentowano w tab. 6.61. Wyraźnie lepsze wyniki osiągnięto dla przeszukiwania z tabu, które osiągnęło skuteczność powyżej 77%. Algorytm mrowiskowy nie przekroczył dla tej wielkości szyfrów 30% skuteczności. W związku z tym dalsze badania zdecydowano się kontynuować tylko dla przeszukiwania z tabu. Badania te wykazały również na przykładzie RC4 (pkt 6.1.4), VMPC (pkt 6.2.4) i RC4+ (pkt 6.3.4) skuteczność przeszukiwania z tabu w kryptoanalizie szyfrów strumieniowych opartych na permutacji.

Tabela 6.60: Porównanie skuteczności przeszukiwania z tabu, algorytmu mrówki wierzchołkowej oraz algorytmu mrowiskowego dla dziesięcioelementowych wersji szyfrów

Algorytm / Szyfr	przeszukiwanie z tabu	algorytm mrówki wierzchołkowej	algorytm mrowiskowy
RC4_10	**100,0%** **141 613** (pkt 6.1.1)	76,9% – (pkt 6.1.2)	**100,0%** 251 778 (pkt 6.1.3)
VMPC_10	**95,3%** (pkt 6.2.1)	68,1% (pkt 6.2.2)	95,6% (pkt 6.2.3)
RC4+_10	**98,8%** (pkt 6.3.1)	64,4% (pkt 6.3.2)	98,4% (pkt 6.3.3)

Tabela 6.61: Porównanie skuteczności przeszukiwania z tabu oraz algorytmu mrowiskowego dla szesnastoelementowych wersji szyfrów

Algorytm / Szyfr	przeszukiwanie z tabu	algorytm mrowiskowy
RC4_16	**98,1%** (pkt 6.1.4)	28,1% (pkt 6.1.5)
VMPC_16	**80,6%** (pkt 6.2.4)	8,1% (pkt 6.2.5)
RC4+_16	**77,5%** (pkt 6.3.4)	6,3% (pkt 6.3.5)

Warto dodać, że wszystkie testy dla dziesięcio- i szesnastoelementowych wersji analizowanych szyfrów, które osiągnęły wartość funkcji dopasowania równą 100%, zwróciły prawidłową permutację. Żaden z testów, które osiągnęły pełne pokrycie strumienia szyfrującego, nie zwrócił permutacji innej niż poszukiwana.

Kolejne eksperymenty dotyczyły analizy pełnych wersji szyfrów strumieniowych RC4 (pkt 6.1.6), VMPC (pkt 6.2.6) i RC4+ (pkt 6.3.6). Na podstawie analizy przebiegu funkcji dopasowania w ograniczonej liczbie iteracji określono, ile permutacji średnio trzeba będzie sprawdzić przed znalezieniem permutacji prawidłowej. Zbiorcze wyniki na ten temat zebrano w tab. 6.62.

Tabela 6.62: Porównanie wyników kryptoanalizy z przeszukiwaniem z tabu uzyskanych dla analizowanych szyfrów

Szyfr	Liczba iteracji		Wielkość iteracji	Rozpatrzonych permutacji	
RC4	2^{84}	$\cdot$	2^{8}	$= 2^{92}$	(pkt 6.1.6)
VMPC	2^{143}	$\cdot$	2^{14}	$= 2^{157}$	(pkt 6.2.6)
RC4+	2^{44}	$\cdot$	2^{8}	$= 2^{52}$	(pkt 6.3.6)

Wyniki uzyskane podczas analizy przeszukiwaniem z tabu dla szyfru RC4 skonfrontowano z innymi metaheurystykami opisanymi w literaturze [Fer13, FO14]: algorytmem genetycznym, symulowanym wyżarzaniem oraz optymalizacją stadną cząsteczek. Udowodniono teoretycznie, że w porównaniu z pozostałymi rozważonymi metaheurystykami przeszukiwanie z tabu wymaga sprawdzenia najmniejszej liczby permutacji (pkt 6.1.7).

Udowodniono teoretycznie, że przeszukiwanie z tabu doprowadzi do ujawnienia prawidłowej permutacji szybciej niż dotychczasowe znane ataki dla szyfru VMPC (pkt 6.2.6). Przedstawiona metoda jest również pierwszym atakiem kryptoanalitycznym na szyfr RC4+, umożliwiającym odtworzenie stanu wewnętrznego bez dodatkowej ingerencji w działanie algorytmu (pkt 6.3.6). Wyniki tych badań zebrano zbiorczo w tab. 6.63. Na koniec udowodniono również, że otrzymane wyniki są dużo lepsze niż można oczekiwać od przypadkowej zgodności bajtów w strumieniu szyfrującym (podrozdz. 6.4).

Badania zostały wykonane według tego samego schematu dla szyfrów RC4, VMPC oraz RC4+. Zaczynając od wersji dziesięcioelementowej i szesnastoelementowej, na których udowodniono skuteczność proponowanej metody, a kończąc na wersjach pełnych, dla których na podstawie zachowania w ograniczonej liczbie iteracji ekstrapolowano dalszy przybliżony przebieg funkcji dopasowania.

Tabela 6.63: Porównanie wyników kryptoanalizy z przeszukiwaniem z tabu z najlepszymi znanymi atakami

Szyfr	Najlepszy znany atak	Przeszukiwanie z tabu
RC4	**atak na WPA-TKIP w czasie godziny** [VP15]	2^{92} (pkt 6.1.6)
VMPC	2^{260} [Żół04]	$\mathbf{2^{157}}$ (pkt 6.2.6)
RC4+	— (tylko analiza błędów [BSK13])	$\mathbf{2^{52}}$ (pkt 6.3.6)

Na podstawie wyników przeprowadzonych badań można wyprowadzić następujące dodatkowe wnioski:

- Ponieważ zaproponowany algorytm pozwolił obniżyć złożoność kryptoanalizy w stosunku do ataków znanych z literatury, dalsze badania w tym kierunku mogą doprowadzić do opracowania ataku czasu rzeczywistego na szyfry strumieniowe: VMPC i RC4+.

- Z przedstawionych badań wynika, że szyfr VMPC ma bardzo dobre właściwości mieszające i jest on bardziej odporny na kryptoanalizę przedstawionym algorytmem niż szyfry RC4 i RC4+.

- Ponieważ proponowany atak ma na celu znalezienie stanu wewnętrznego szyfru, jego złożoność będzie taka sama niezależnie od wielkości klucza. Innymi słowy, jeśli długość klucza zostanie zwiększona przykładowo ze 128 bitów do 256 bitów, kryptoanaliza z przeszukiwaniem z tabu nie będzie wymagała więcej czasu ani większej liczby operacji.

Rozdział 7

Zakończenie

Rozprawa poświęcona jest zagadnieniom związanym z użyciem algorytmów metaheurystycznych w kryptoanalizie. Celem rozprawy było opracowanie autorskich wersji wybranych algorytmów metaheurystycznych przystosowanych do kryptoanalizy szyfrów strumieniowych. W ramach realizacji celu w rozprawie przeprowadzono analizę istniejących szyfrów strumieniowych (podrozdz. 2.4) oraz sformułowano problem ataku kryptoanalitycznego (podrozdz. 2.3). Opracowano autorskie wersje algorytmów metaheurystycznych przystosowane do kryptoanalizy szyfrów strumieniowych opartych na permutacji (rozdz. 4-5).

Na potrzeby badań eksperymentalnych zaimplementowano w języku C# narzędzie umożliwiające automatyczną analizę zadanego strumienia szyfrującego z użyciem opracowanych algorytmów. Efektywność oprogramowania sprawdzono przy użyciu środowiska Visual Studio, w szczególności za pomocą narzędzia profilującego (Performance Profiler). Do analizy wyników użyto arkusza kalkulacyjnego, a wykresy wykonano z użyciem pakietu TikZ. W części eksperymentalnej sprawdzono jakość wyników otrzymanych przez proponowane algorytmy (rozdz. 6) pod kątem liczby testów zakończonych sukcesem oraz liczby sprawdzonych w tych testach permutacji dla mniejszych wersji szyfrów, a także pod kątem wzrostu wartości funkcji dopasowania w kolejnych iteracjach dla pełnych wersji szyfrów.

Proponowane algorytmy kryptoanalizy zostały poddane testowaniu z zastosowaniem trzech szyfrów strumieniowych: RC4 (pkt 2.4.1), VMPC (pkt 2.4.2) i RC4+ (pkt 2.4.3). Dla każdego z nich użyto różnych kluczy dobranych tak, by charakteryzowały się różnorodną strukturą (tab. 6.18, 6.38, 6.57). Umożliwiło to opracowanie ataku niezależnego od właściwości klucza, a także od konstrukcji samego szyfru, zakładając, że stan wewnętrzny szyfru może być reprezentowany jako permutacja liczb z zadanego zakresu.

Dla wszystkich zaproponowanych algorytmów zbadano cztery rodzaje funkcji dopasowania, z czego trzy to funkcje autorskie: bajtowe z sąsiadami, ważone i ważone odwrotnie (podrozdz. 4.2). Autorskie funkcje dopasowania umożliwiły uzyskanie lepszych wyników dla wszystkich trzech analizowanych algorytmów metaheurystycznych (przeszukiwanie z tabu tab. 6.41, algorytm mrówki wierzchołkowej tab. 6.6, 6.26, 6.45 oraz algorytm mrowiskowy tab. 6.11, 6.31, 6.50) w porównaniu z funkcją znaną z literatury [Fer13, FO14].

W przypadku algorytmu przeszukiwania z tabu dokonano analizy czterech rodzajów otoczenia – różnice osiąganych wyników między nimi były bardzo wyraźne. Najlepsze wyniki osiągnięto dla otoczenia „połowa par" (tab. 6.23) oraz „elementy losowo" (tab. 6.3, 6.42). Możliwe, że pozostałe dwa rodzaje otoczenia powodowały ugrzęźnięcie algorytmu w optimach lokalnych. Dokonano analizy również sześciu wartości horyzontu, z czego trzy z nich o charakterze autorskim (tab. 6.4, 6.24, 6.43). Dla szyfru RC4+ zaproponowany horyzont $\log_2 |R(r)|$ (gdzie $|R(r)|$ to rozmiar otoczenia rozwiązania r) pozwolił na uzyskanie wyników lepszych niż dla pozostałych analizowanych wartości horyzontu (tab. 6.43). Sprawdzono również wersje algorytmu przeszukiwania z tabu z zastosowaniem kryterium aspiracji lub z jego pominięciem. Choć różnice nie były duże, to dla wszystkich trzech szyfrów najlepsze wyniki osiągnięto bez stosowania kryterium aspiracji (tab. 6.5, 6.25, 6.44).

W ramach badań testowano również zaproponowany algorytm mrówki wierzchołkowej. W przeciwieństwie do klasycznych algorytmów optymalizacji mrowiskowej, w badanym algorytmie przyjęto, że feromon jest odkładany w wierzchołkach grafu, a nie na jego krawędziach. Zaproponowano uzależnienie wartości maksymalnej feromonu τ_{max} odłożonego na mapie feromonowej od wielkości rozpatrywanej permutacji (tab. 6.7, 6.27, 6.46). Najlepsze wyniki osiągnięto dla najmniejszej z rozpatrzonych wartości feromonu maksymalnego $\tau_{max} = 2|S|$ (gdzie $|S|$ jest wielkością analizowanej permutacji). W przypadku gdy feromon maksymalny nie został określony ($\tau_{max} = \infty$) znacząca większość eksperymentów zakończyła się niepowodzeniem. W wyniku dodatniego sprzężenia zwrotnego nastąpiła tutaj silna akumulacja feromonu na jednej ze ścieżek, przez co mrówki utkwiły w optimum lokalnym. Z trzech rozważonych wartości feromonu domyślnego τ_0, najlepsze okazały się wartości skrajne: $\tau_0 = 1$ (tab. 6.8, 6.28) lub $\tau_0 = \tau_{max}$ (tab. 6.47). Parametr ρ określający szybkość odparowania feromonu najlepiej sprawdził się z wartością najmniejszą z rozpatrywanych, tj. $\rho = 1$ (tab. 6.9, 6.29, 6.48). Wraz z algorytmem mrówki wierzchołkowej wprowadzono również trzy autorskie rodzaje budowy rozwiązań w postaci permutacji przystosowanych do rozważanego w rozprawie problemu. Były to: budowa rozwiązań od początku, od końca i losowo. W zależności od szyfru poddanego kryptoanalizie, inny rodzaj budowy rozwiązań sprawdził się najlepiej (dla szyfru

RC4 – losowo, tab. 6.10; dla szyfru VMPC – od początku, tab. 6.30; dla szyfru RC4+ – od końca, tab. 6.49).

W rozprawie zbadano także algorytm mrowiskowy w zastosowaniu do kryptoanalizy szyfrów strumieniowych. Oryginalnym podejściem było zastosowanie grafu, którego wierzchołki reprezentowały pary elementów podlegających zamianie w danej permutacji (rys. 5.1, podrozdz. 5.2). Największy wpływ na jakość osiąganych wyników miał parametr β związany ze względną ważnością informacji feromonowej w stosunku do informacji heurystycznej. Wyraźnie lepsze wyniki otrzymano w przypadku, gdy informacja heurystyczna miała znaczną przewagę nad informacją feromonową (tab. 6.13, 6.33, 6.52). Podobny wpływ na wyniki miał odpowiedni dobór parametru q_0 określający zależność między eksploracją a eksploatacją (tab. 6.14, 6.34, 6.53). Z badań wynika, że im większy był nacisk na eksplorację (czyli im mniejsza wartość parametru q_0), tym lepsze były uzyskiwane wyniki. Mniejszy wpływ na jakość otrzymywanych wyników miały pozostałe parametry, tj. τ_0 (tab. 6.12, 6.32, 6.51) określające wartość domyślną śladu feromonowego oraz parametr ρ (tab. 6.15, 6.35, 6.54) związany z odparowaniem śladu feromonowego.

W rozprawie testowano każdy z zaproponowanych algorytmów pod kątem wpływu wartości parametrów na osiągane przez nie wyniki, a także porównano ze sobą wyniki osiągnięte przez wszystkie trzy algorytmy. Już dla małych, dziesięcioelementowych wersji szyfrów wyniki dla algorytmu mrówki wierzchołkowej były wyraźnie gorsze od wyników pozostałych dwóch algorytmów (tab. 6.60). W rozważanym problemie kryptoanalitycznym nie da się określić informacji heurystycznej, która mogłaby mieć wpływ na wybór poszczególnych kroków w algorytmie mrówki wierzchołkowej (pkt 5.1.2). Mrówki korzystają jedynie z informacji w postaci śladu feromonowego. W wyniku tego następowała niekorzystna zmiana początkowo równomiernego rozkładu prawdopodobieństwa wybrania danego węzła, w rozkład nierównomierny, faworyzujący specyficzny rejon przestrzeni rozwiązań, co powodowało szybką zbieżność do optimum lokalnego. W rezultacie, intensywna eksploracja odbywała się w początkowych iteracjach algorytmu, a następnie ulegała osłabieniu wraz z kolejnymi iteracjami. Wyniki dla algorytmu mrowiskowego, w którym parametr β określający wielkość śladu feromonowego wyraźnie wpływał na zbieżność algorytmu, potwierdzają to spostrzeżenie. Ponadto w algorytmie mrówki wierzchołkowej nie ma możliwości budowania częściowych rozwiązań i ich oceny w trakcie budowania rozwiązań. To stanowi przyczynę, jak się wydaje, niepowodzenia algorytmu mrówki wierzchołkowej w rozwiązywaniu problemu kryptoanalizy szyfrów strumieniowych.

Na dalszym etapie badań eksperymentalnych pominięto zatem algorytm mrówki wierzchołkowej i skupiono się na algorytmie przeszukiwania z tabu oraz algorytmie mrowiskowym. Badania te wykonano na szesnastoelementowych wersjach szyfrów.

Analizując ich wyniki zauważono, że algorytm przeszukiwania z tabu okazał się lepszy niż algorytm mrowiskowy (tab. 6.61), zarówno pod względem liczby testów zakończonych sukcesem, jak i liczby sprawdzonych permutacji w tych testach. Tego algorytmu użyto zatem do badań zasadniczych na pełnych 256-bajtowych wersjach analizowanych szyfrów strumieniowych.

Wyniki uzyskane podczas kryptoanalizy pełnej wersji szyfru RC4 z użyciem przeszukiwania z tabu porównano z wynikami uzyskanymi przez inne algorytmy metaheurystyczne [Fer13, FO14], wykazując, że przeszukiwanie z tabu prowadzi do uzyskania wyników lepszych niż one (pkt 6.1.7). Po sprawdzeniu tej samej liczby permutacji przeszukiwanie z tabu pozwoliło osiągnąć wartość funkcji dopasowania o 6,2% wyższą niż w przypadku algorytmu genetycznego, o 23,9% wyższą niż w przypadku symulowanego wyżarzania oraz o 26,2% wyższą niż w przypadku optymalizacji stadnej cząsteczek (tab. 6.20). Sprawdzenie wszystkich możliwych permutacji początkowych to ok. 2^{1684} możliwości do przeanalizowania. W stosunku do najlepszych wyników otrzymanych przez algorytm genetyczny, gdzie potrzebnych było sprawdzenie 2^{124} permutacji w celu odnalezienia poszukiwanej permutacji, użycie przeszukiwania z tabu pozwoliło zmniejszyć tę liczbę do 2^{92} sprawdzonych permutacji.

Rezultaty otrzymane przy użyciu algorytmu przeszukiwania z tabu porównano także z istniejącymi atakami na wybrane szyfry strumieniowe (pkt. 2.4.1-3). Dla szyfrów VMPC i RC4+ proponowany algorytm kryptoanalizy z użyciem przeszukiwania z tabu wydaje się być lepszy niż dotychczas znane algorytmy kryptoanalizy (tab. 6.63). W przypadku szyfru VMPC przestrzeń możliwych kluczy to 2^{512}, a najlepszy znany z literatury algorytm ataku wymaga 2^{260} operacji obliczeniowych [Żół04] – proponowany atak przy użyciu algorytmu przeszukiwania z tabu wymaga sprawdzenia średnio 2^{157} permutacji. W przypadku szyfru RC4+ znany jest tylko atak wymagający ingerencji w działający szyfr [BSK13], a liczba możliwych permutacji to ok. 2^{1684} możliwości – proponowany w rozprawie atak wymaga sprawdzenia średnio 2^{52} permutacji bez konieczności ingerencji w szyfr.

Mając na uwadze powyższe, rezultaty niniejszej rozprawy wzbogacają portfolio ataków kryptoanalitycznych o nową metodę ataku oraz mobilizują do jeszcze bardziej szczegółowej weryfikacji używanych i projektowanych szyfrów. Analiza wyników przeprowadzonych eksperymentów oraz poczynione obserwacje mogą stanowić podstawę do dalszych prac nad kryptoanalizą szyfrów strumieniowych przy wykorzystaniu algorytmów metaheurystycznych.

Podsumowując omówione powyżej wyniki uzyskane w ramach rozprawy należy stwierdzić, że teza rozprawy została wykazana, czyli algorytmy metaheurystyczne stanowią skuteczne narzędzie kryptoanalityczne do łamania współczesnych szyfrów strumieniowych. Spośród zaproponowanych i zbadanych eksperymentalnie algorytmów metaheurystycznych,

najlepsze wyniki uzyskano przy użyciu algorytmu przeszukiwania z tabu. Pozostałe rozważane algorytmy metaheurystyczne dawały co prawda gorsze wyniki, aczkolwiek nie należy wykluczyć, że dalsze ich ulepszanie mogłoby te wyniki poprawić.

Również dodatkowe cele, jakie postawiono w rozprawie, zostały osiągnięte. W szczególności:

- przeanalizowano algorytmy metaheurystyczne pod kątem możliwości ich zastosowania w kryptoanalizie (rozdz. 3),
- przeanalizowano istniejące metody kryptoanalizy wybranych szyfrów strumieniowych: RC4, VMPC i RC4+ (pkt. 2.4.1-3),
- zaimplementowano w języku C# szyfry strumieniowe RC4, VMPC i RC4+, a także opracowano w języku XML bazę ich wektorów testowych (rozdz. 6),
- zaprojektowano i zaimplementowano w języku C# algorytm przeszukiwania z tabu oraz wybrane algorytmy optymalizacji mrowiskowej, umożliwiając automatyczną kryptoanalizę zadanego strumienia szyfrującego (rozdz. 6),
- zbadano zaproponowane algorytmy pod kątem wpływu wartości parametrów na jakość uzyskiwanych rozwiązań (pkt. 6.1.1-3, 6.2.1-3, 6.3.1-3; podrozdz. 6.5),
- wykonano eksperymenty na mniejszych wersjach szyfrów w celu wybrania najbardziej skutecznego algorytmu do dalszych badań (pkt. 6.1.1-5, 6.2.1-5, 6.3.1-5),
- porównano algorytm przeszukiwania z tabu z innymi metaheurystykami pod kątem kryptoanalizy (pkt 6.1.7) oraz z atakami znanymi z literatury na wybrane szyfry strumieniowe (pkt. 6.1.6, 6.2.6, 6.3.6; podrozdz. 6.5).

Dalsze prace nad rozwijaniem badanych zagadnień mogą przyjąć wiele różnych kierunków. Przykładowo interesujące byłoby sprawdzenie następujących możliwości:

1. Przeprowadzenie kryptoanalizy za pomocą przedstawionego algorytmu szyfru strumieniowego RC4A [PP04], który charakteryzuje się podobną budową. Podobnie jak szyfry RC4, VMPC i RC4+, szyfr RC4A w każdym cyklu generuje jeden bajt strumienia szyfrującego. Zasadniczą różnicą w stosunku do rozpatrywanych w niniejszej rozprawie szyfrów jest bardziej rozbudowany stan wewnętrzny szyfru RC4A, który stanowią dwie niezależne tablice permutacji.
2. Weryfikacja odporności (lub jej braku) na kryptoanalizę z algorytmem przeszukiwania z tabu szyfrów strumieniowych o zupełnie innej konstrukcji, np. szyfru HC-128 [Wu08] z portfolio projektu eSTREAM [ECR04]. Stan wewnętrzny tego szyfru

stanowią dwie tablice składające się z 512 32-bitowych elementów. Nie przyjmuje się żadnych założeń odnośnie wzajemnej relacji między tymi elementami, w szczególności nie są to permutacje kolejnych liczb naturalnych. W każdym cyklu szyfr HC-128 generuje jedno 32-bitowe słowo strumienia szyfrującego.

3. Hybrydowe połączenie zaprezentowanego w niniejszej rozprawie algorytmu kryptoanalizy z inną ze znanych metaheurystyk, bądź z istniejącymi w literaturze atakami o charakterze klasycznym. Odkryte do tej pory dla szyfrów RC4, VMPC i RC4+ statystyczne odchylenia od ciągów prawdziwie losowych mogłyby stanowić dodatkową informację pozwalającą na bardziej efektywne przeszukiwanie przestrzeni możliwych stanów wewnętrznych.

4. Modyfikacja algorytmu optymalizacji mrowiskowej w celu zwiększenia jego skuteczności w kryptoanalizie szyfrów strumieniowych. Można rozważyć dynamiczną zmianę parametrów, przykładowo zmieniając liczbę mrówek w kolejnych iteracjach lub sterując stopniem eksploracji oraz eksploatacji w zależności od jakości bieżących rozwiązań.

Bibliografia

[AAE14] A. E. Amin i A. Abd Elbadea. Building Cryptosystem Based on Genetic Algorithm and Pattern Recognition Concepts for Academic Institution. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(10):128–139, 2014.

[AAs04] H. Ali i M. Al-salami. Timing Attack Prospect for RSA Cryptanalysts Using Genetic Algorithm Technique. *The International Arab Journal of Information Technology*, 1(1):80–84, 2004.

[ABK98] R. Anderson, E. Biham, i L. Knudsen. Serpent: A Proposal for the Advanced Encryption Standard. NIST AES Proposal, 1998.

[ABP+13] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, i J. C. N. Schuldt. On the Security of RC4 in TLS. W *Proceedings of the 22 USENIX Conference on Security*, SEC'13, s. 305–320, Berkeley, CA, USA, 2013. USENIX Association.

[AE17] E. Antal i M. Eliáš. Evolutionary Computation in Cryptanalysis of Classical Ciphers. *Tatra Mountains Mathematical Publications*, 70(1):179 – 197, 2017.

[AKAE08] A. Almarimi, A. Kumar, I. Almerhag, i N. Elzoghbi. A New Approach for Data Encryption Using Genetic Algorithms. W *ACIT'2008: Proceedings of the 9th International Arab Conference on Information Technology*, Hammamet, Tunezja, 2008.

[AM14] R. Afarin i S. Mozaffari. Gray Level Image Encryption. *International Journal of Computer, Information, Systems and Control Engineering*, 8(6):865–869, 2014.

[Ant17] E. Antal. *Modern Cryptanalysis of Classical Ciphers (po słowacku)*. Praca doktorska, Slovak University of Technology, Bratysława, Słowacja, 2017.

[AS12] F. Abazari i B. Sadeghian. Cryptanalysis with Ternary Difference: Applied to Block Cipher PRESENT. *International Journal of Information and Electronics Engineering*, 2(3):441–445, 2012.

[ASMR16] S. Amic, K. M. S. Soyjaudah, H. Mohabeer, i G. Ramsawock. Cryptanalysis of DES-16 using Binary Firefly Algorithm. W *2016 IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech)*, s. 94–99, 2016.

[AVR14] R. Asthana, N. Verma, i R. Ratan. Generation of Boolean functions using Genetic Algorithm for cryptographic applications. W *Advance Computing Conference (IACC), 2014 IEEE International*, s. 1361–1366, 2014.

[Awa11] W. S. Awad. Designing Stream Cipher Systems Using Genetic Programming. W C. A. Coello, red., *Learning and Intelligent Optimization*, t. 6683, *Lecture Notes in Computer Science*, s. 308–320. Springer Berlin Heidelberg, 2011.

[Bag96] A. J. Bagnall. The Applications of Genetic Algorithms in Cryptanalysis. Praca magisterska, School of Information Systems, University of East Anglia, 1996.

[BBCS15] A. K. Bhateja, A. Bhateja, S. Chaudhury, i P. K. Saxena. Cryptanalysis of Vigenere cipher using Cuckoo Search. *Applied Soft Computing*, 26(0):315–324, 2015.

[BD13] M. Banerjee i D. Das. Cyclic Cryptographic Technique Using Substitution and Genetic Function. *International Journal of Emerging Technology and Advanced Engineering*, 3(2):508–515, 2013.

[BD14a] U. Boryczka i K. Dworak. Cryptanalysis of Transposition Cipher Using Evolutionary Algorithms. W D. Hwang, J. J. Jung, i N. T. Nguyen, red., *Computational Collective Intelligence. Technologies and Applications*, t. 8733, *Lecture Notes in Computer Science*, s. 623–632. Springer International Publishing, 2014.

[BD14b] U. Boryczka i K. Dworak. Genetic Transformation Techniques in Cryptanalysis. W N. T. Nguyen, B. Attachoo, B. Trawiński, i K. Somboonviwat, red., *Intelligent Information and Database Systems*, t. 8398, *Lecture Notes in Computer Science*, s. 147–156. Springer International Publishing, 2014.

[BGK77] D. K. Branstad, J. Gait, i S. Katzke. Report on the workshop on cryptography in support of computer security. Raport techniczny, National Bureau of Standards, 1977.

[Bha14] A. Bhateja. Analysis of Different Cryptosystems Using Meta-Heuristic Techniques. *IEEE International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, s. 1931–1934, 2014.

[BHS97] B. Bullnheimer, R. F. Hartl, i Ch. Strauß. A New Rank Based Version of the Ant System – A Computational Study. *Central European Journal for Operations Research and Economics*, 7:25–38, 1997.

[BKL+07] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, i C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. W P. Paillier i I. Verbauwhede, red., *Cryptographic Hardware and Embedded Systems – CHES 2007*, t. 4727, *Lecture Notes in Computer Science*, s. 450–466. Springer Berlin Heidelberg, 2007.

[BKS05] P. Bouvry, G. Klein, i F. Seredynski. Weak Key Analysis and Micro-controller Implementation of CA Stream Ciphers. W R. Khosla, R. J. Howlett, i L. C. Jain, red., *Knowledge-Based Intelligent Information and Engineering Systems*, s. 910–915, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[BO99] G. Barbarosoglu i D. Ozgur. A tabu search algorithm for the vehicle routing problem. *Computers & Operations Research*, 26(3):255–270, 1999.

[BP09] U. Boryczka i I. Polak. Cumulation of Pheromone Values in Web Searching Algorithm. W K. A. Cyran, S. Kozielski, J. F. Peters, U. Stańczyk, i A. Wakulicz-Deja, red., *Man-Machine Interactions*, s. 515–522, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[BR03] Ch. Blum i A. Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.

[BSK13] S. Banik, S. Sarkar, i R. Kacker. Security Analysis of the RC4+ Stream Cipher. W G. Paul i S. Vaudenay, red., *INDOCRYPT*, t. 8250, *Lecture Notes in Computer Science*, s. 297–307. Springer, 2013.

[BSS08] A. G. Bafghi, R. Safabakhsh, i B. Sadeghiyan. Finding the differential characteristics of block ciphers with neural networks. *Information Sciences*, 178(15):3118–3132, 2008.

[Car10] C. Carlet. Boolean functions for cryptography and error-correcting codes. W *Boolean models and methods in mathematics, computer science, and engineering*, s. 257–397. Cambridge: Cambridge University Press, 2010.

[CdVHM00] O. Cordón, I. Fernández de Viana, F. Herrera, i L. Moreno. A New ACO Model Integrating Evolutionary Computation Concepts: The Best-Worst Ant System. W *Proceedings of ANTS 2000 – From Ant Colonies to Artificial Ants: Third International Workshop on Ant Algorithms*, s. 22–29, 2000.

[Čer85] V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

[CH97] D. Costa i A. Hertz. Ants can colour graphs. *Journal of the Operational Research Society*, 48(3):295–305, 1997.

[Cla03] J. A. Clark. Nature-inspired cryptography: past, present and future. W *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, t. 3, s. 1647–1654, 2003.

[Coh95] F. Cohen. A Short History of Cryptography, 1995. http://all.net/edu/curr/ip/Chap2-1.html.

[CR88] B. Chor i R. L. Rivest. A knapsack-type public key cryptosystem based on arithmetic in finite fields. *Information Theory, IEEE Transactions on*, 34(5):901–909, 1988.

[Cyp94] RC4 Source Code. Cypherpunks, 1994. http://cypherpunks.venona.com/archive/1994/09/msg00304.html.

[DB15] K. Dworak i U. Boryczka. Cryptanalysis of SDES Using Modified Version of Binary Particle Swarm Optimization. W M. Núñez, N. T. Nguyen, D. Camacho, i B. Trawiński, red., *Computational Collective Intelligence*, s. 159–168, Cham, 2015. Springer International Publishing.

[DB16] K. Dworak i U. Boryczka. Differential Cryptanalysis of FEAL4 Using Evolutionary Algorithm. W N. T. Nguyen, L. Iliadis, Y. Manolopoulos, i B. Trawiński, red., *Computational Collective Intelligence*, s. 102–112, Cham, 2016. Springer International Publishing.

[DB17] K. Dworak i U. Boryczka. Genetic Algorithm as Optimization Tool for Differential Cryptanalysis of DES6. W N. T. Nguyen, G. A. Papadopoulos,

P. Jędrzejowicz, B. Trawiński, i G. Vossen, red., *Computational Collective Intelligence*, s. 107–116, Cham, 2017. Springer International Publishing.

[DCD99] G. Di Caro i M. Dorigo. AntNet: Distributed Stigmergetic Control for Communications Networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1999.

[DDC99] M. Dorigo i G. Di Caro. New Ideas in Optimization. W D. Corne, M. Dorigo, . Glover, D. Dasgupta, P. Moscato, R. Poli, i K. V. Price, red., *New Ideas in Optimization*, rozdz. The Ant Colony Optimization Meta-heuristic, s. 11–32. McGraw-Hill Ltd., UK, Maidenhead, UK, Anglia, 1999.

[DG97] M. Dorigo i L. M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[DG03a] A. Dimovski i D. Gligoroski. Attacks on the Transposition Ciphers using Optimization Heuristics. W *Proceedings of ICEST 2003*, s. 1–4, Sofia, Bułgaria, 2003.

[DG03b] A. Dimovski i D. Gligoroski. Generating highly nonlinear Boolean functions using a genetic algorithm. W *Telecommunications in Modern Satellite, Cable and Broadcasting Service, 2003. TELSIKS 2003. 6th International Conference on*, t. 2, s. 604–607, 2003.

[DMC91] M. Dorigo, V. Maniezzo, i A. Colorni. Positive Feedback as a Search Strategy. Raport techniczny, Politecnico di Milano, Włochy, 1991.

[DNBK16] K. Dworak, J. Nalepa, U. Boryczka, i M. Kawulok. *Cryptanalysis of SDES Using Genetic and Memetic Algorithms*, s. 3–14. Springer International Publishing, Cham, 2016.

[Dor92] M. Dorigo. *Optimization, Learning and Natural Algorithms (po włosku)*. Praca doktorska, Dipartimento di Elettronica, Politecnico di Milano, Mediolan, Włochy, 1992.

[DR99] J. Daemen i V. Rijmen. AES Proposal: RIJNDAEL. AES submission, 1999.

[DS03] M. Dorigo i T. Stützle. *The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances*, s. 250–285. Springer US, Boston, MA, 2003.

[Dwo14] K. Dworak. Zastosowanie algorytmów ewolucyjnych w kryptoanalizie klasycznej. *Systemy Inteligencji Obliczeniowej*, s. 145–153, 2014.

[ECR04] ECRYPT. eSTREAM, 2004. http://www.ecrypt.eu.org/stream/.

[EMP+76] W. F. Ehrsam, C. H. W. Meyer, R. L. Powers, J. L. Smith, i W. L. Tuchman. Product Block Cipher for Data Security, 1976. US Patent 3,962,539.

[ETS94] European digital cellular telecommunications system (Phase 2); Security related network functions (GSM 03.20). ETSI – Raport techniczny, 1994.

[ETS96] Security Algorithms Group of Experts (SAGE); Report on the specification and evaluation of the GSM cipher algorithm A5/2. ETSI – Raport techniczny, 1996.

[Fei73] H. Feistel. Cryptography and Computer Privacy. *Scientific American*, 228(5):15–23, 1973.

[Fei74] H. Feistel. Block cipher cryptographic system, 1974. US Patent 3,798,359.

[Fer13] B. N. Ferriman. Cryptanalysis of the RC4 Stream Cipher using Evolutionary Computation Methods. Praca magisterska, The University of Guelph, Guelph, Ontario, Kanada, 2013.

[FO14] B. Ferriman i Ch. Obimbo. Solving for the RC4 stream cipher state register using a genetic algorithm. *International Journal of Advanced Computer Science and Applications*, 5(5):218–223, 2014.

[Gar09] P. Garg. Genetic algorithms, tabu search and simulated annealing: a comparison between three approaches for the cryptanalysis of transposition cipher. *Journal of Theoretical and Applied Information Technology*, s. 387–392, 2009.

[Gar10] P. Garg. A Comparison between Memetic algorithm and Genetic algorithm for the cryptanalysis of Simplified Data Encryption Standard algorithm. *CoRR*, abs/1004.0574, 2010.

[GK03] F.W. Glover i G.A. Kochenberger. *Handbook of Metaheuristics*. International Series in Operations Research & Management Science. Springer US, 2003.

[Glo86] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Comput. Oper. Res.*, 13(5):533–549, 1986.

[GM86] F. Glover i C. McMillan. The general employee scheduling problem. An integration of MS and AI. *Computers & Operations Research*, 13(5):563 – 573, 1986.

[GMPS12] S. Gupta, S. Maitra, G. Paul, i S. Sarkar. (Non-)Random Sequences from (Non-)Random Permutations – Analysis of RC4 Stream Cipher. *Journal of Cryptology*, 27(1):67–108, 2012.

[GMS04] C. Guéret, N. Monmarché, i Mohamed Slimane. Ants Can Play Music. W M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, i T. Stützle, red., *Ant Colony Optimization and Swarm Intelligence*, s. 310–317, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[GS07] P. Garg i A. Shastri. An Improved Cryptanalytic Attack on Knapsack Cipher using Genetic Algorithm. *International Journal of Information Technology*, 3:553–560, 2007.

[GSS93] M. Gendreau, P. Soriano, i L. Salvail. Solving the maximum clique problem using a tabu search approach. *Annals of Operations Research*, 41(4):385–403, 1993.

[GTA99] L. M. Gambardella, E. Taillard, i G. Agazzi. MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows. Raport techniczny, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, 1999.

[GY12] R. Goyal i S. P. Yadav. An evolutionary approach to construct cryptographically strong Boolean functions. *Int J Syst Assur Eng Manag*, 3(1):1–5, 2012.

[Har06] B. Harris. Improved Arcfour Modes for the Secure Shell (SSH) Transport Layer Protocol. Networking Working Group – Request for Comments: 4345, 2006. http://tools.ietf.org/html/rfc4345.

[HBH+07] H. M. H. Husein, B. I. Bayoumi, F. S. Holail, B. E. M. Hasan, i M. Z. A. El-Mageed. A Genetic Algorithm for Cryptanalysis of DES-8. *I. J. Network Security*, 5(2):213–219, 2007.

[HdW87] A. Hertz i D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987.

[HGDM+11] G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede, i J. Vandewalle. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1(4):293–302, 2011.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, USA, 1975.

[HSH13] M. Heydari, G. L. Shabgahi, i M. M. Heydari. Cryptanalysis of Transposition Ciphers with Long Key Lengths Using an Improved Genetic Algorithm. *World Applied Sciences Journal*, 21(8):1194–1199, 2013.

[Hu10] W. Hu. Cryptanalysis of TEA Using Quantum-Inspired Genetic Algorithms. *JSEA*, 3(1):50–57, 2010.

[IOWM14] T. Isobe, T. Ohigashi, Y. Watanabe, i M. Morii. Full Plaintext Recovery Attack on Broadcast RC4. W S. Moriai, red., *Fast Software Encryption*, s. 179–202, Springer Berlin Heidelberg, 2014.

[IR08] P. Itaim i M. C. Riff. Applying Differential Cryptanalysis for XTEA using a Genetic Algorithm, 2008.

[JMB13] S. Jhajharia, S. Mishra, i S. Bali. Public Key Cryptography Using Particle Swarm Optimization and Genetic Algorithms. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(6):832–839, 2013.

[JQDCJA09] A. Jevtić, J. Quintanilla-Dominguez, M. G. Cortina-Januchs, i D. Andina. Edge detection using ant colony search algorithm and multiscale contrast enhancement. W *2009 IEEE International Conference on Systems, Man and Cybernetics*, s. 2193–2198, 2009.

[KAD13] S. Khan, A. Ali, i M. Y. Durrani. Ant-Crypto, a Cryptographer for Data Encryption Standard. *IJCSI International Journal of Computer Science Issues*, 10(1):400–406, 2013.

[Kah04] D. Kahn. *Łamacze kodów: historia kryptologii*. TAO Tajemnica, Atak, Obrona. Wydawnictwa Naukowo-Techniczne, 2004.

[KB12] F. U. Khan i S. Bhatia. A Novel Approach to Genetic Algorithm Based Cryptography. *International Journal of Research in Computer Science*, 2(3):7–10, 2012.

[KB16] J. Kozak i U. Boryczka. Collective data mining in the ant colony decision tree approach. *Information Sciences*, 372:126 – 147, 2016.

[KD08] M. Kumral i R. Dimitrakoponlos. Selection of waste dump sites using a tabu search algorithm. *Journal of the Southern African Institute of Mining and Metallurgy*, 108:9–13, 2008.

[KE95] J. Kennedy i R. C. Eberhart. Particle swarm optimization. W *Proceedings of the IEEE International Conference on Neural Networks*, s. 1942–1948, 1995.

[Ker83] A. Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5–83, 1883.

[KGV83] S. Kirkpatrick, C. D. Gelatt, i M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[Kie18] A. Kielijan. Zastosowanie PSO w kryptoanalizie. Praca inżynierska, Uniwersytet Śląski, Katowice, 2018.

[KK05a] P. Kotlarz i Z. Kotulski. Metody sztucznej inteligencji we współczesnej kryptografii. W *XI Konferencja użytkowników i deweloperów ORACLE*, s. 179–189, Kościelisko, 2005.

[KK05b] P. Kotlarz i Z. Kotulski. On Application of Neural Networks for S-Boxes Design. W P. S. Szczepaniak, J. Kacprzyk, i A. Niewiadomski, red., *Advances in Web Intelligence*, t. 3528, *Lecture Notes in Computer Science*, s. 243–248. Springer Berlin Heidelberg, 2005.

[KK06] P. Kotlarz i Z. Kotulski. Neuronowy układ szyfrujący – analiza bezpieczeństwa. W *XII Konferencja użytkowników i deweloperów ORACLE*, s. 227–235, Kościelisko, 2006.

[KK07a] P. Kotlarz i Z. Kotulski. Neural Network as a Programmable Block Cipher. W J. Pejaś i K. Saeed, red., *Advances in Information Processing and Protection*, s. 241–250. Springer US, 2007.

[KK07b] P. Kotlarz i Z. Kotulski. Przekształcenia szyfrujące – realizacje neuronowe, ocena jakości algorytmu szyfrującego. *XIII Konferencja użytkowników i deweloperów ORACLE*, s. 224–233, 2007.

[KK08] P. Kotlarz i Z. Kotulski. Neuronowa realizacja nieliniowych przekształceń szyfrujących. W *Telecommunication Review – Telecommunication News*, t. 81(77), s. 1293–1303. Stowarzyszenie Elektryków Polskich, 2008.

[KMP97] J. Kołodziejczyk, J. Miller, i P. Phillips. The application of genetic algorithm in cryptoanalysis of knapsack cipher. W *Proceedings of Fourth International*

Conference Pattern Recognition and Information Processing, t. 1, s. 394–401, 1997.

[Kno94] J. Knox. Tabu Search Performance on the Symmetric Traveling Salesman Problem. *Comput. Oper. Res.*, 21(8):867–876, 1994.

[Kob87] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[Kob06] N. Koblitz. *Wykład z teorii liczb i kryptografii*. Wydawnictwa Naukowo-Techniczne, Warszawa, Wydanie 2, 2006.

[Kot08] P. Kotlarz. *Sieci neuronowe we wspomaganiu rozwiązywania problemów kryptologii*. Praca doktorska, Instytut Podstawowych Problemów Techniki PAN, Warszawa, 2008.

[Kul06] K. Kulesza. On inverting the VMPC one-way function, 2006.

[Las18] G. Lasry. *A Methodology for the Cryptanalysis of Classical Ciphers with Search Metaheuristics*. Kassel University Press, 2018.

[LG93] M. Laguna i F. Glover. Bandwidth Packing: A Tabu Search Approach. *Management Science*, 39(4):492–500, 1993.

[LHZW12] S. Li, Y. Hu, Y. Zhao, i Y. Wang. Improved cryptanalysis of the VMPC stream cipher. *Journal of Computational Information Systems*, 8(2):831–838, 2012.

[LK95] F. Lin i C. Kao. A genetic algorithm for ciphertext-only attack in cryptanalysis. W *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*, t. 1, s. 650–654, 1995.

[LKW16] G. Lasry, N. Kopal, i A. Wacker. Cryptanalysis of Columnar Transposition Cipher with Long Keys. *Cryptologia*, 40(4):374–398, 2016.

[LMS+07] E.C. Laskari, G.C. Meletiou, Y.C. Stamatiou, D.K. Tasoulis, i M.N. Vrahatis. Assessing the effectiveness of artificial neural networks on problems related to elliptic curve cryptography. *Mathematical and Computer Modelling*, 46(1–2):174 – 179, 2007.

[LMSV07a] E.C. Laskari, G.C. Meletiou, Y.C. Stamatiou, i M.N. Vrahatis. Applying evolutionary computation methods for the cryptanalysis of Feistel ciphers. *Applied Mathematics and Computation*, 184(1):63 – 72, 2007.

[LMSV07b] E.C. Laskari, G.C. Meletiou, Y.C. Stamatiou, i M.N. Vrahatis. Cryptography and Cryptanalysis Through Computational Intelligence. W N. Nedjah, A. Abraham, i L. M. Mourelle, red., *Computational Intelligence in Information Assurance and Security*, t. 57, *Studies in Computational Intelligence*, s. 1–49. Springer Berlin Heidelberg, 2007.

[LNKW17] G. Lasry, I. Niebel, N. Kopal, i A. Wacker. Deciphering ADFGVX messages from the Eastern Front of World War I. *Cryptologia*, 41(2):101–136, 2017.

[LP11] J. Luthra i S. K. Pal. A hybrid Firefly Algorithm using genetic operators for the cryptanalysis of a monoalphabetic substitution cipher. W *2011 World Congress on Information and Communication Technologies*, s. 202–206, 2011.

[Mat03] R. Matoušek. Knapsack Cipher and Cryptanalyst using Heuristic Methods, 2003.

[Max05] A. Maximov. *Two Linear Distinguishing Attacks on VMPC and RC4A and Weakness of RC4 Family of Stream Ciphers*, s. 342–358. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[MBC+99] W. Millan, L. Burnett, G. Carter, A. Clark, i E. Dawson. Evolutionary Heuristics for Finding Cryptographically Strong S-Boxes. W V. Varadharajan i Y. Mu, red., *Information and Communication Security*, t. 1726, *Lecture Notes in Computer Science*, s. 263–274. Springer Berlin Heidelberg, 1999.

[MC99] V. Maniezzo i A. Colorni. The Ant System Applied to the Quadratic Assignment Problem. *IEEE Trans. on Knowl. and Data Eng.*, 11(5):769–778, 1999.

[MCD98] W. Millan, A. Clark, i E. Dawson. Heuristic design of cryptographically strong balanced Boolean functions. W K. Nyberg, red., *Advances in Cryptology – EUROCRYPT'98*, t. 1403, *Lecture Notes in Computer Science*, s. 489–499. Springer Berlin Heidelberg, 1998.

[McL08] J. McLaughlin. *Chapter 8 – cryptanalysis with combinatorial optimisation*. University of York, 2008.

[McL12] J. D. McLaughlin. *Applications of search techniques to cryptanalysis and the construction of cipher components*. Praca doktorska, University of York, Department of Computer Science, 2012.

[MH78] R. Merkle i M. Hellman. Hiding Information and Signatures in Trapdoor Knapsacks. *IEEE Trans. Inf. Theor.*, 24(5):525–530, 1978.

[MHM17] M. A. Mosa, A. Hamouda, i M. Marei. Graph coloring and ACO based summarization for social networks. *Expert Systems with Applications*, 74:115 – 126, 2017.

[Mic06] Z. Michalewicz. *Jak to rozwiązać, czyli Nowoczesna heurystyka*. Wydawnictwa Naukowo-Techniczne, Warszawa, 2006.

[Mil86] V. S. Miller. Use of Elliptic Curves in Cryptography. W H. C. Williams, red., *Advances in Cryptology – CRYPTO '85 Proceedings*, t. 218, *Lecture Notes in Computer Science*, s. 417–426. Springer Berlin Heidelberg, 1986.

[Mir02] I. Mironov. (Not So) Random Shuffles of RC4. W M. Yung, red., *Advances in Cryptology — CRYPTO 2002*, s. 304–319, Springer Berlin, Heidelberg, 2002.

[Mit97] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., Nowy Jork, NY, USA, Wydanie 1, 1997.

[MM14] T. Mekhaznia i M. E. B. Menai. Cryptanalysis of Classical Ciphers with Ant Algorithms. *Int. J. Metaheuristics*, 3(3):175–198, 2014.

[MMS02] D. Merkle, M. Middendorf, i H. Schmeck. Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, 6(4):333–346, 2002.

[MNS+13] S. K. Mahata, S. Nogaja, S. Srivastava, M. Dey, i S. Som. A Novel Approach to Cryptography using Modified Substitution Cipher and Hybrid Crossover Technique. *IJCA Proceedings on Computing Communication and Sensor Network 2013*, CCSN 2013(2):33–37, 2013.

[Mos89] P. Moscato. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Raport techniczny C3P Report 826, California Institute of Technology, 1989.

[MP43] W. S. McCulloch i W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5(4):115–133, 1943.

[MP08] S. Maitra i G. Paul. *Analysis of RC4 and Proposal of Additional Layers for Better Security Margin*, s. 27–39. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[MPSJ17] P. K. Mudgal, R. Purohit, R. Sharma, i M. K. Jangir. Application of genetic algorithm in cryptanalysis of mono-alphabetic substitution cipher. W *2017*

International Conference on Computing, Communication and Automation (ICCCA), s. 400–405, 2017.

[MS17] S. A. Moses i W. Sangplung. Resource planning for just-in-time make-to-order environments: A scalable methodology using tabu search. *Cogent Engineering*, 4(1):1341289, 2017.

[MVOV05] A. J. Menezes, P. C. Van Oorschot, i S. A. Vanstone. *Kryptografia stosowana*. Warszawa : Wydawnictwa Naukowo-Techniczne, 2005.

[MVR09] R. Muthuregunathan, D. Venkataraman, i P. Rajasekaran. Cryptanalysis of Knapsack Cipher using Parallel Evolutionary Computing. *International Journal of Recent Trends in Engineering*, 1(1):260–263, 2009.

[MW06] R. Morelli i R. Walde. Evolving Keys for Periodic Polyalphabetic Ciphers. W G. Sutcliffe i R. Goebel, red., *FLAIRS Conference*, s. 445–450. AAAI Press, 2006.

[Naj10] M. Najjar. Generating of homogeneous Boolean functions with high nonlinearity using genetic algorithm. *International Journal of Computer Science and Network Security*, 10(3):266–273, 2010.

[NIS01] Advanced Encryption Standard (AES). FIPS PUB 197, 2001. National Institute of Standards and Technology, U.S. Department of Commerce, Federal Information Processing Standards Publication 197.

[NN14] D. P. G. Naik i G. R. Naik. Asymmetric Key Encryption using Genetic Algorithm. *International Journal of Latest Trends in Engineering and Technology*, 3(3), 2014.

[NPK13] D. Nagde, R. Patel, i D. Kelde. New Approach for Data Encryption using Two Way Crossover. *International Journal of Computer Science and Information Technologies*, 4(1):58–60, 2013.

[NR04] N. Nalini i G. R. Rao. A New Encryption And Decryption Algorithm Combining The Features Of Genetic Algorithm (GA) And Cryptography. W *International Conference on Cognitive Systems, New Delhi*, 2004.

[NR05] N. Nalini i G. R. Rao. Cryptanalysis of Simplified Data Encryption Standard via Optimization Heuristics. W *Intelligent Sensing and Information Processing, 2005. ICISIP 2005. Third International Conference on*, s. 74–79, 2005.

[NR06] N. Nalini i G. R. Rao. Experiments on Cryptanalysing Block Ciphers via Evolutionary Computation Paradigms. W *Proceedings of the 7th WSEAS International Conference on Evolutionary Computing, Cavtat, Croatia*, s. 20–26, 2006.

[NW97] R. M. Needham i D. J. Wheeler. Tea extensions. Raport techniczny, University of Cambridge, 1997.

[Odl90] A. M. Odlyzko. The rise and fall of knapsack cryptosystems. W *Cryptology and Computational Number Theory*, s. 75–88. A.M.S, 1990.

[Pas02] K. M. Passino. Biomimicry of Bacterial Foraging for distributed optimization and control. *IEEE Control Systems Magazine*, 22(3):52–67, 2002.

[PB13] I. Polak i M. Boryczka. Breaking LFSR Using Genetic Algorithm. W Costin Badica, Ngoc Thanh Nguyen, i Marius Brezovan, red., *ICCCI*, t. 8083, *Lecture Notes in Computer Science*, s. 731–738. Springer, 2013.

[PB14a] I. Polak i M. Boryczka. Algorytmy inspirowane naturą w kryptoanalizie. *Studia Bezpieczeństwa Narodowego, Kryptologia i Cyberbezpieczeństwo*, 6:185–197, 2014.

[PB14b] I. Polak i M. Boryczka. Kryptoanaliza A5/1 i A5/2 przy użyciu algorytmów genetycznych. *Systemy Inteligencji Obliczeniowej*, s. 145–153, 2014.

[PB15] I. Polak i M. Boryczka. Genetic Algorithm in Stream Cipher Cryptanalysis. W Manuel Núñez, Ngoc Thanh Nguyen, David Camacho, i Bogdan Trawiński, red., *Computational Collective Intelligence*, t. 9330, *Lecture Notes in Computer Science*, s. 149–158. Springer International Publishing, 2015.

[PB18] I. Polak i M. Boryczka. Tabu search against permutation based stream ciphers. *International Journal of Electronics and Telecommunications*, 64(2):137–145, 2018.

[PH78] S. Pohlig i M. Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.

[PLF02] R. S. Parpinelli, H. S. Lopes, i A. A. Freitas. Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 6(4):321–332, 2002.

[PM18] E. Papenhausen i K. Mueller. Coding Ants – Optimization of {GPU} Code Using Ant Colony Optimization. *Computer Languages, Systems & Structures*, 2018.

[Pol17] I. Polak. Tabu search against permutation based stream ciphers (Referat). W *17-th Central European Conference on Cryptology*, Warszawa, Polska, 2017.

[Pol18] I. Polak. Tabu search for VMPC stream cipher cryptanalysis (Referat). W *18-th Central European Conference on Cryptology*, Smolenice, Słowacja, 2018.

[POR+05] D. T. Pham, S. Otri, S. Rahim, A. Ghanbarzadeh, E. Koc, i M. Zaidi. The Bees Algorithm. Raport techniczny, Manufacturing Engineering Centre, Cardiff University, UK, 2005.

[PP04] S. Paul i B. Preneel. *A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher*, s. 245–259. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[PPS15] K. G. Paterson, B. Poettering, i J. C. N. Schuldt. Plaintext Recovery Attacks Against WPA/TKIP. W C. Cid i C. Rechberger, red., *Fast Software Encryption*, s. 325–349, Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[PSM+11] S. Palit, S.N. Sinha, M.A. Molla, A. Khanra, i M. Kule. A cryptanalytic attack on the knapsack cryptosystem using binary Firefly algorithm. W *Computer and Communication Technology (ICCCT), 2011 2nd International Conference on*, s. 428–432, 2011.

[RB11] D. R. G. Ramani i L. Balasubramanian. Genetic Algorithm solution for Cryptanalysis of Knapsack Cipher with Knapsack Sequence of Size 16. *International Journal of Computer Applications*, 35(11):17–23, 2011.

[RCS03] M. Russell, J. A. Clark, i S. Stepney. Making the most of two heuristics: breaking transposition ciphers with ants. W *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, t. 4, s. 2653–2658, 2003.

[RHKB17] A. Rybarczyk, A. Hertz, M. Kasprzak, i J. Blazewicz. Tabu search for the RNA partial degradation problem. *International Journal of Applied Mathematics and Computer Science*, 27(2):401–415, 2017.

[RK11] G. N. Rajendra i B. R. Kaur. A New Approach for Data Encryption Using Genetic Algorithms and Brain Mu Waves. *International Journal Of Scientific And Engineering Research*, 2(5):87–91, 2011.

[RS12] Rajashekarappa i K. M. S. Soyjaudah. Cryptanalysis of Simplified-Data Encryption Standard Using Tabu Search Method. W K. R. Venugopal i L. M. Patnaik, red., *Wireless Networks and Computational Intelligence*, s. 561–568, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[RSA78] R. L. Rivest, A. Shamir, i L. M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[SA13a] P. Saveetha i S. Arumugam. Privacy Preservation on Stream Ciphers Using Genetic Algorithm with Pseudo-Random Series. *Australian Journal of Basic and Applied Sciences*, 7(12):1–8, 2013.

[SA13b] A. Soni i S. Agrawal. Key Generation Using Genetic Algorithm for Image Encryption. *International Journal of Computer Science and Mobile Computing*, 2(6):376–383, 2013.

[Sai14] T. Saini. One Time Password Generator System. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(3):781–785, 2014.

[Sar15] S. Sarkar. Further non-randomness in RC4, RC4A and VMPC. *Cryptography and Communications*, 7(3):317–330, 2015.

[Sch94] B. Schneier. Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish). W *Fast Software Encryption, Cambridge Security Workshop*, s. 191–204, Springer-Verlag, Londyn, UK, 1994.

[Sch96] E. F. Schaefer. A Simplified Data Encryption Standard algorithm. *Cryptologia*, 20(1):77–84, 1996.

[Sch02] B. Schneier. *Kryptografia dla praktyków: Protokoły, algorytmy i programy źródłowe w języku C*. Wydawnictwa Naukowo-Techniczne, Warszawa, Wydanie 2, 2002.

[SH00] T. Stützle i H. H. Hoos. MAX–MIN Ant System. *Future Generation Computer Systems*, 16(8):889 – 914, 2000.

[SH05] A. Shmygelska i H. H. Hoos. An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem. *BMC Bioinformatics*, 6(1):30, 2005.

[Sim17] G. J. Simmons. Cryptology. W *Encyclopædia Britannica*. Encyclopædia Britannica, inc., 2017.

[Sin01] S. Singh. *Księga szyfrów*. Albatros, Warszawa, 2001.

[SJ11] J. Strombergson i S. Josefsson. Test Vectors for the Stream Cipher RC4. Internet Engineering Task Force (IETF) – Request for Comments: 6229, 2011. http://tools.ietf.org/html/rfc6229.

[SM88] A. Shimizu i S. Miyaguchi. Fast Data Encipherment Algorithm FEAL. W D. Chaum i Wyn L. Price, red., *Advances in Cryptology — EUROCRYPT' 87*, s. 267–278, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.

[SM14] A. Sarkar i J. K. Mandal. Computational science guided soft computing based cryptographic technique using ant colony intelligence for wireless communication (ACICT). *International Journal on Computational Sciences & Applications (IJCSA)*, 4(5):61–72, 2014.

[SP94] M. Srinivas i L. M. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(4):656–667, 1994.

[SP97] R. Storn i K. Price. Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *J. of Global Optimization*, 11(4):341–359, 1997.

[SP08] N. K. Sreelaja i G. A. V. Pai. Swarm intelligence based key generation for text encryption in cellular networks. W *Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on*, s. 622–629, 2008.

[SP12a] G. Selvi i T. Purusothaman. Cryptanalysis of Simple Block Ciphers using Extensive Heuristic Attacks. *European Journal of Scientific Research*, 78(2):198–221, 2012.

[SP12b] N. K. Sreelaja i G. A. Vijayalakshmi Pai. Stream cipher for binary image encryption using Ant Colony Optimization based key generation. *Applied Soft Computing*, 12(9):2879 – 2895, 2012.

[Spi93] R. Spillman. Cryptanalysis of knapsack ciphers using genetic algorithms. *Cryptologia*, 17(4):367–377, 1993.

[SPM+11] S. N. Sinha, S. Palit, M. A. Molla, A. Khanra, i M. Kule. A cryptanalytic attack on Knapsack cipher using Differential Evolution algorithm. W *Recent Advances in Intelligent Computational Systems (RAICS), 2011 IEEE*, s. 317–320, 2011.

[SS11a] M. Szaban i F. Seredynski. Designing Cryptographically Strong S-Boxes with Use of 1D Cellular Automata. *Journal of Cellular Automata*, 6:91–104, 2011.

[SS11b] M. Szaban i F. Seredynski. Improving quality of DES S-boxes by cellular automata-based S-boxes. *The Journal of Supercomputing*, 57(2):216–226, 2011.

[SSB06] M. Szaban, F. Seredynski, i P. Bouvry. Evolving collective behavior of cellular automata for cryptography. W *MELECON 2006 - 2006 IEEE Mediterranean Electrotechnical Conference*, s. 799–802, 2006.

[Sta12] W. Stallings. *Kryptografia i bezpieczeństwo sieci komputerowych: matematyka szyfrów i techniki kryptologii*. Helion, Gliwice, 2012.

[SVP09] N. K. Sreelaja i G. A. Vijayalakshmi Pai. Design of Stream Cipher for Text Encryption using Particle Swarm Optimization based Key Generation. *Journal of Information Assurance and Security*, 4:30–41, 2009.

[SYWZ08] J. Song, F. Yang, M. Wang, i H. Zhang. Cryptanalysis of Transposition Cipher Using Simulated Annealing Genetic Algorithm. W L. Kang, Z. Cai, X. Yan, i Y. Liu, red., *Advances in Computation and Intelligence*, t. 5370, *Lecture Notes in Computer Science*, s. 795–802. Springer Berlin Heidelberg, 2008.

[SZMW07] J. Song, H. Zhang, Q. Meng, i Z. Wang. Cryptanalysis of Four-Round DES Based on Genetic Algorithm. W *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on*, s. 2326–2329, 2007.

[TSK+05] Y. Tsunoo, T. Saito, H. Kubo, M. Shigeri, T. Suzaki, i T. Kawabata. The Most Efficient Distinguishing Attack on VMPC and RC4A, 2005.

[UY06] M. F. Uddin i A. M. Youssef. Cryptanalysis of Simple Substitution Ciphers Using Particle Swarm Optimization. W *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, s. 677–680, 2006.

[vN51] J. von Neumann. The general and logical theory of automata. W *Cerebral Mechanisms in Behavior. The Hixon Symposium*, s. 1–41. J. Wiley & Sons, Inc., Nowy Jork, N. Y.; Chapman & Hall, Ltd., Londyn, 1951.

[vN66] J. von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.

[VP15] M. Vanhoef i F. Piessens. All Your Biases Belong to Us: Breaking RC4 in WPA-TKIP and TLS. W *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, s. 97–112, Berkeley, CA, USA, 2015.

[WN94] D. J. Wheeler i R. M. Needham. TEA, a Tiny Encryption Algorithm. W B. Preneel, red., *FSE*, t. 1008, *Lecture Notes in Computer Science*, s. 363–366. Springer, 1994.

[Wu08] H. Wu. *The Stream Cipher HC-128*, s. 39–47. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[WW11] P. Wicher i K. Wilkosz. Wykorzystanie algorytmu Tabu Search do lokalizacji baterii kondensatorów w sieci elektroenergetycznej. *Acta Energetica*, nr 2:67–73, 2011.

[YAA13] H. A. Younes, W. S. Awad, i A. A. Abd. Attacking of Stream Cipher Systems Using a Genetic Algorithm. *Journal of University of Thi-Qar*, 8(3):78–84, 2013.

[Yan08] X. Yang. *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2008.

[YD09] X. Yang i S. Deb. Cuckoo Search via Lévy Flights. W *NaBIC*, s. 210–214. IEEE, 2009.

[You00] H. A. Younes. Attacking stream cipher systems using genetic algorithm. Praca magisterska, Department of Computer Science, Basrah University. Basra, Irak, 2000.

[YS98] I. F. T. Yaseen i H. V. Sahasrabuddhe. Breaking multiplicative knapsack ciphers using a genetic algorithm. W *Proceedings of the International Conference on Knowledge Based Computer Systems*, s. 129–139, 1998.

[YSZ08] F. Yang, J. Song, i H. Zhang. Quantitative Cryptanalysis of Six-Round DES Using Evolutionary Algorithms. W L. Kang, Z. Cai, X. Yan, i Y. Liu, red., *Advances in Computation and Intelligence*, t. 5370, *Lecture Notes in Computer Science*, s. 134–141. Springer Berlin Heidelberg, 2008.

[Żół04] B. Żółtak. VMPC One-Way Function and Stream Cipher. W B. Roy i W. Meier, red., *Fast Software Encryption*, t. 3017, *Lecture Notes in Computer Science*, s. 210–225. Springer Berlin Heidelberg, 2004.

[ZS02] H. Zhang i G. Sun. Feature selection using tabu search method. *Pattern Recognition*, 35(3):701 – 711, 2002.

Spis algorytmów

Spis rysunków

Spis tabel